Multitouch Interface Programming

The development of computer user interfaces has generally moved from interaction that was unnatural to more naturalistic means. Using a command line to manipulate a file, for example, is a rather abstract means of doing so, and mouse based direct-manipulation systems brought this action closer to the natural experience. However, even using a mouse as a tool to carry out interaction is an experience unknown to humankind until just recently. Multitouch technology brings computer interaction one step closer to what is familiar to us: using our entire hands to manipulate the objects we see in front of us. Users can engage both hands and multiple fingers to affect the images they see on screen in ways that are natural, though sometimes physically impossible, as if their fingers were performing a magical spell on the virtual objects.

While multitouch interfaces have been used in research projects for some time, they are only now starting to be incorporated into commercial products. This paper will explore a bit of the past, present, and future of multitouch systems. While by no means an exhaustive review, I will discuss some history of multitouch research, present some of the multitouch devices and software systems available today, and explore how a multitouch application would be programmed on an event driven system like the Apple iPhone. Finally, I will look briefly at some of the future possibilities that are enabled by multitouch technology.

Multitouch Background

A multitouch system is one where multiple points of spatial input can be provided to a computer. While some systems allow for these multiple spatial inputs to be provided by several conventional pointing devices used simultaneously, I will be focusing on touch-based systems. The fingers themselves can either be placed on the screen itself, directly manipulating the images, or on another offscreen surface like a tablet or touchpad. The multiple touches may
come from a single hand, two hands, and may even come from multiple users simultaneously. Actual input can be pointing type actions utilizing a single finger, but can also be multi-finger chorded gestures that can encode other actions.

Just as the mouse had existed in research settings many years before the Apple Macintosh introduced it to people’s homes, user interface researchers have been exploring the possibilities of multitouch for over 20 years. As early as 1982, a device had been developed that allowed multiple pressure points to be detected by a computer. Dubbed the “Flexible Machine Interface”, the device consisted of a material with properties such that any touch would cause that area of the backside of the material to turn a darker shade. A camera placed behind the material could detect these changes in color (Nimish, 1982, from Buxton, 2008). In a 1983 book titled Artificial Reality, Myron Krueger described a rich gestural system for interacting with a computer unencumbered by artificial input devices or limitations (Buxton, 2008). It is from this work that the original two-fingered pinching motion to zoom in and out was first introduced, though it may be more well known from a vision-of-the-future type film presented by Bruce Tognazinni of Sun Microsystems in 1992. By 1984, the first multitouch display was shown by Bell Labs (Buxton, 2008). For a more detailed history, see Bill Buxton’s website.

While multitouch technology was known to the research community, the public was given a vision of rich gestural multitouch interaction in the 2002 thriller Minority Report. In the film, the main character uses special gloves with lights in them to interact with multiple streams of video on a large wall display. The character is shown manipulating images through pinching motions as well as using a two-handed wiping gesture to clear away his workspace. This image was so strong for some that many news reports describing multitouch systems often begin by announcing that the Minority Report technology is now available.

Multitouch Devices and Software Currently Available

One of the most high profile applications of multitouch technology has been in the Apple iPhone and its relatives. The device uses a capacitive touch display, capable of distinguishing multiple points of input. Software included with the device recognizes multi-finger gestures like pinching to enable users to navigate full size web pages on the smaller screen. Two-finger flicks can be used to scroll through contact lists or albums (Apple Inc., 2008a). While the device has been available for nearly a year, third-party developers are officially limited to creating
applications that run in the web browser, with no native support for multitouch events. This hasn’t stopped hackers from reverse engineering the system to program native applications, but it does mean that documentation is limited. This should change sometime in the coming months with the announced release of the iPhone Software Developer’s Kit.

Apple has also been integrating multitouch technology into the touchpads of laptops. The newest release, the MacBook Air, has a larger touchpad allowing users to use some multi-finger gestures in Apple applications, like pinching to resize in iPhoto and using a three-finger swipe to move forward and back in the Safari browser (Apple Inc., 2008b). A two-finger scrolling gesture has been available in Apple laptops for several years.

In 2006, Jeff Han of NYU demonstrated a multitouch display and some basic applications at the 2006 Technology, Education and Design (TED) conference. Later videos of his demonstrations were uploaded to YouTube and viewed millions of times. The demonstrations show very fluid interactions, but seem to be limited to the few sample programs created, rather than being general-purpose devices. The touchscreen system uses a method called Frustrated Total Internal Reflection (FTIR), in which a plastic panel is illuminated internally by LEDs in a way that none of the light can escape, and an image projected through the rear. When a finger is placed on the surface, the total internal reflection effect is broken in that spot, creating a bright white blob on the rear of the panel. A camera behind the panel can report these spots back to a computer (Han, 2005). These systems allow for multiple finger and multiple user input, though may not be able to discriminate which fingers belong to which users.

The DiamondTouch from Mitsubishi Electric Research Laboratory solves the problem of distinguishing between multiple users in an interesting way. The system consists of a projector mounted above a table with a capacitive touch surface. Users sit around the table in chairs that are connected to the system, such that a unique RF signal is conducted through the chairs and through the fingers of each user. This provides a unique signature for every user that can be sensed by the system and interpreted by a computer (Dietz & Leigh, 2001).

Other hardware for performing multitouch input is available as well. Microsoft has announced a tabletop multitouch system named Microsoft Surface designed for multiuser interaction at restaurants, casinos, or at home (Microsoft, 2008). The system allows users to interact with a table full of virtual photos using the pinching and dragging gestures to move them around and resize them. There are also some homebrew approaches to creating multitouch
hardware, like Johnny Chung Lee’s tutorial (2008) on using the Wii Remote to create a system very much like the Minority Report interface, or the $2 dyeSight bag of water and dye technique (Ellingsen, 2007).

As multitouch hardware becomes more prevalent, software systems are being developed to take advantage of them on the desktop. The Multi-Pointer X server is a modification of the X Window server that allows multiple pointers, either from traditional input devices or multitouch panels, to interact with multitouch aware software (Hutterer & Thomas, 2007). The system also allows for legacy support by sending normal mouse events to non-multipointer aware applications. A project called “touchlib” allows multitouch input devices to be connected to Windows PCs, and allows events to be sent through a standard protocol to software like PureData or Processing (White Noise Audio Software, 2008).

**Programming Multitouch Systems**

There are several software systems that enable multitouch interaction, with many differences between them. I will present the system that is used by the Apple iPhone and iPod Touch, which is an event based system programmed in Objective-C. As I mentioned previously, there is no official documentation for this system yet, and limited unofficial support. Much of the information presented here is outlined in *iPhone Open Application Development* by Jonathan Zdziarski, currently available online and, shortly, in print. As I don’t actually have a device with which to test any of these claims, I am largely trusting the material I can find online and my own analysis of available source code.

Regardless of device, multitouch events can be delivered in three forms, depending on the level of processing the application is required to do. The lowest-level event is known as a blob event. This consists of a bitmap of the areas of touch activation. The application, then, is required to filter out noise and analyze the bitmap to find the locations of fingers. Next, a system can deliver discrete touch events. These consist of a list of coordinate of each touch location. An application that is interested in using gestures must then translate the coordinates into the proper gestural action. A system could also deliver gestural events, telling the application that, for example, a 20% pinching motion was registered. Though I have seen very little explicit support for these gestural events, I imagine that as gestures become more popular and more
pervasive, operating systems will provide these standards to prevent each application from redoing the interpretive work.

The iPhone works using discrete touch events, and distinguishes between single touch and multitouch events. Single touch events are handled in the same way as mouseEvents, using mouseDown, mouseUp, and mouseDragged events to indicate a single finger’s actions. While present in the API, the mouseMoved, mouseEntered, and mouseExited events don’t make sense in the context of a touchscreen, and so aren’t used. Each of these events carries a screen coordinate with it so that the application can respond.

Multitouch events invoke different methods when they are fired, and can only be used from subclasses of the UIView base class. Multitouch events correlate with single touch events, as shown in the table below. A two finger gesture starts as a single touch, and so the sequence of events for a gesture would be: mouseDown when the first finger touches the surface, gestureStarted when the second finger is pressed, gestureChanged when either finger moves, gestureEnded when a finger is lifted, and mouseUp as the second finger leaves.

<table>
<thead>
<tr>
<th>Single Touch</th>
<th>Multitouch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finger is added</td>
<td>- (void) gestureStarted: (</td>
</tr>
<tr>
<td></td>
<td>____GSEvent *) event</td>
</tr>
<tr>
<td>Finger is removed</td>
<td>- (void) gestureEnded: (</td>
</tr>
<tr>
<td></td>
<td>____GSEvent *) event</td>
</tr>
<tr>
<td>Finger is dragged</td>
<td>- (void) mouseDragged: (</td>
</tr>
<tr>
<td></td>
<td>____GSEvent *) event</td>
</tr>
<tr>
<td></td>
<td>- (void) gestureChanged: (</td>
</tr>
<tr>
<td></td>
<td>____GSEvent *) event</td>
</tr>
</tbody>
</table>

Table 1: Correlation between single touch and multitouch events.

When a multitouch event is received, there are two ways to extract the location information. One method is to use calls to GSEventGetInnerMostPathPosition and GSEventGetOuterMostPathPosition. These calls will provide the leftmost and rightmost finger positions, regardless of the number of fingers on the screen at a given time. This can be useful for implementing the pinching and rotating gestures common to multitouch interaction. If more points are needed, each GEvent object contains an array of touch points, and an application can iterate through these to retrieve as many as are needed.

Some gesture events can also be used by building off of base classes that already respond to certain gestures. For example, the UIScroller class already contains the functionality to do
scrolling in many directions. By inheriting from this class and overriding the method that handles scrolling, a crude scroll event handler can be constructed.

Future Possibilities for Multitouch

Multitouch technology allows some interesting ways to interact with computers. Systems like the Microsoft Surface and Mitsubishi DiamondTouch enable many people to gather around a table-top system to work together. While this is an interesting deviation from traditional computer interaction, the applications shown so far, like many people looking through virtual pictures, have yet to show great improvements over the capabilities of single-user systems.

Multitouch technology eases interaction with a research idea known as the Zooming User Interface or ZUI. Just as Google Earth presents vast degrees of detail, but also allows for a zoomed out overview, a ZUI can do the same for a file system. This has been shown in projects like Pad++ (Bederson & Hollan, 1994) and the more recent effort, Piccolo (Bederson, et al., 2004). The two finger pinch provides an ideal means of navigating a zooming environment.

Also of interest is a recent patent filing from Apple Inc. showing a dictionary of multitouch gestures that can be invoked to present an onscreen menu of multitouch movements (Topolsky 2007). This may suggest Apple’s intention to make more use of multitouch gestures on desktop systems. Gestures could be used more quickly than menu selections by removing the pointing time required to open a menu, and if learned, could be used without consulting the onscreen dictionary at all, yielding even faster interaction.

Conclusion

Multitouch technology is an exciting way to interact with machines, and while previously in the domain of science fiction, is now becoming available to consumers. Multitouch applications work well with the existing event driven model that drives GUI interaction, and affords new possibilities that have yet to be realized. As of yet, though, there haven’t been more than a couple of toy applications that really take advantage of the technology. It will be interesting to see what is the killer app for multitouch displays, beyond sorting photographs and viewing maps. Perhaps as infinite streams of video are accessible over the internet, we will see an interface out of sci-fi allowing us to navigate through it. Regardless, we are bound to see greater implementations and applications of multitouch technology in the near future.
Works Cited


