

Implementation of Touchscreen Overlay with Gestural Input

Dennis Wey
3/2/2010

Abstract

Adding a touchscreen to your design will enhance user experience, allowing the user to manipulate interactive objects in a natural way. Further improvements can be realized by adding gesture recognition which lets the user draw symbols on-screen to activate predefined shortcuts, saving them time and effort. Finally, the use of global gestures will enable users who are visually impaired to utilize the existing touchscreen functionality without having to use third-party devices and software.

Introduction

The essence of “Universal Design” is enabling all users to use a device intuitively and effectively. Allowing unencumbered access to both users with or without disabilities is a crucial design factor in any consumer product. By using a touchscreen, we simplify UI by interlinking tactile input with system output. With gestures, we effectively decouple the UI from its physical location onscreen and enable visually impaired users to use the touchscreen as well. Furthermore, using gesture shortcuts allows both visually impaired and non-visually impaired users faster access, cutting the amount of time spent navigating through a complex menu structure.

Objectives

The purpose of this document is to introduce the touchscreen overlay and installation process, and motivate the touch gesture concept and its implementation. In this particular solution, the overlay will be used with a small 8.9” screen laptop computer, and the gesture software will be implemented in Visual Basic.

Touchscreen Overlay

Using a touchscreen allows you to provide users the ability to interact directly with displayed information. Doing so allows you to eliminate the need for intermediate devices such as a keyboard or mouse, saving room and allowing for more compact design solutions.

One way of providing this functionality is to use a transparent touch device and put it over a LCD monitor. In this example, a Hoda 8.9” touchscreen overlay kit will be used with an ASUS EEE netbook PC.

A. Hardware Installation

First, the netbook must be dismantled. Start by removing the screws on the bottom in order to gain access to the screen assembly. Note that some plastic tabs may be broken in the process of doing this; be sure that you are willing to void your warranty and potentially damage the cosmetic appearance of your netbook. Be aware that certain models of netbook are easier to dismantle than others, and that newer ones may present more trouble.

Once the backing has been removed, the screen assembly can be removed as well in a similar fashion.

For the following steps, be sure that you are grounded. The internals of the laptop as well as the touchscreen controller are sensitive to electrostatic discharge, and you could potentially damage these components. When you have removed the screen, you will notice a cable connecting the built-in

camera/microphone at the top of the screen assembly. Disconnect this and replace it with the USB controller hub provided with the kit. If needed, reconnect these components to the other USB 2.0 ports on the hub. This allows the touchscreen to transmit data to the PC and draw power directly. After the hub is connected, it needs to be affixed to the motherboard. Using double-sided tape, attach it to the lower right-hand corner of the motherboard.

Next, apply some more double-sided tape to the LCD screen to prepare it for the overlay attachment. Foam padding is also provided that can be put in between these components to allow a better fit. After the touchscreen is in place, connect the ribbon cable to the USB controller hub, making sure to snake it along the LCD screen cable. Try and position the ribbon cable as to prevent any pinching that could occur when opening and closing the lid.

Finally, replace the keyboard and the plastic chasis, and replace all the screws.

B. Calibration

Once the computer is reassembled, the touchscreen should automatically be detected as an input device. If the installation has been done correctly, it should respond to touches and click where you point the device. Note, however, it may not correspond exactly to your input, or may even be reversed. In order to correct for this, the screen must be calibrated. The CD provided with the kit will allow you to run a calibration program which consists of pressing down on all four corners until calibration data is collected. This can be run once or on boot.

Gestures

A. Gesture Concept

Gestures are interpreted user-input that can call various commands. By recognizing simple drawn symbols, it is possible to create a library of gesture inputs associated with various commands or menu choices.

Much like keyboard shortcuts, such as ctrl+C and ctrl+V for copy and paste, input device gestures can be used to expediently execute specific commands and circumvent point-and-click menu structures. As one can imagine, having the ability to perform a repetitive task without having to navigate through convoluted menus constantly saves time and effort. Additionally, since blind users are at a disadvantage using traditional point-and-click menu structures, gestures can be an intuitive and immediately useful way of providing a better interface overall.

B. Implementation

The first step is to collect the user input. Depending on the programming environment being used, there is a function that takes mouse position when called. The method in which mouse position is determined may vary depending on your programming environment, but since we only need to consider pixel values relative to each other it should not cause a discrepancy.

Once the input is collected, it must be filtered. User input will often contain undesired motion artifacts from involuntary muscle movement. By limiting the sample rate, we can eliminate this user “noise”. The desired rate depends on the complexity of your gesture symbol library, but in general a sample rate

below 25 Hz should be good enough. In VB, you can use a timer function to set the sample rate. The following code snippet demonstrates one such implementation:

```
Private Sub Form1_Load1() Handles Me.Load
    gestureCapTimer.Interval = 40 'defines timer tick interval in ms
End Sub

Private Sub Timer1_Tick() Handles gestureCapTimer.Tick
    Dim LocalMousePosition As Point
    LocalMousePosition = Me.PointToClient(Cursor.Position)
    If capCounter < UBound(capturePoints) Then
        capturePoints(capCounter) = LocalMousePosition
        capCounter = capCounter + 1
    End If
End Sub
```

After filtering out unwanted points, we need to calculate the vector between each capture point. There are various ways to do this, but a simple solution is to simply compare movement in the x-direction versus the y-direction, and discard the smaller value. In VB, this can be done with the following code snippet:

```
Public Sub vectorizePoints()

    Dim xDiff As Integer = 0
    Dim yDiff As Integer = 0
    Dim item As Vector 'type structure defined with magnitude, direction
    Dim gVector As ArrayList = New ArrayList 'store computed vectors
    Dim capLength = Array.IndexOf(capturePoints, Nothing)

    For i = 0 To (capLength - 1)
        If capLength >= i + 2 Then
            xDiff = capturePoints(i + 1).X - capturePoints(i).X
            yDiff = capturePoints(i + 1).Y - capturePoints(i).Y

            If System.Math.Abs(xDiff) > System.Math.Abs(yDiff) Then
                item.Direction = "X"
                item.Magnitude = xDiff
                gVector.Add(item)
            ElseIf System.Math.Abs(xDiff) < System.Math.Abs(yDiff) Then
                item.Direction = "Y"
                item.Magnitude = yDiff
                gVector.Add(item)
            End If
        End If
    Next

End Sub
```

This may be unsatisfactory if you require a more complex gesture solution. It does not account for diagonal input or relative lengths (magnitude value used for filtering purposes only). More advanced image recognition algorithms can be used instead if necessary, such as those used in OCR.

Once the list of vectors is captured, the gesture must be matched with predefined symbols. Using the above algorithm, we can simplify the list by looking for consecutive movements in the same direction and joining them together. With this simplified list, the program can iterate through the library and attempt to match the captured data with a predefined symbol. If the first-pass yields no result, then the vector with the shortest length is removed, and the matching process is repeated. In order to prevent a false match, the percentage of vectors removed is tracked. If this increases over a certain threshold, the matching process automatically fails.

C. Example Gesture Algorithm



Fig. 1 – User input

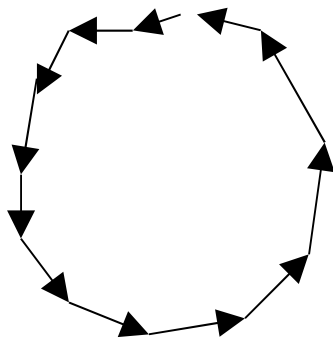


*Fig. 2 –
Captured/filtered
pointer data*

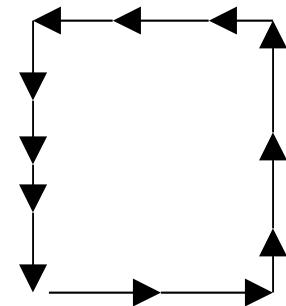


*Fig. 3 – After
vectorization*

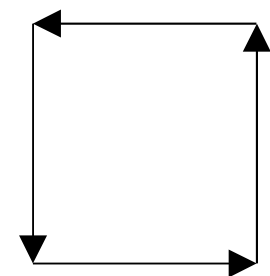


*Fig. 4 – Final
matching result*

Conclusion

By using a touchscreen it is possible to make a user interface that works well for all users. Combining the user's physical interaction with the screen output allows them to interact in an intuitive fashion, eliminating the need for excessive prompting or learning curve. For those with visual impairment as well as those without, the addition of software-based gesture shortcuts augment the touchscreens ability to respond quickly and expediently to user commands.

References

http://www.design.ncsu.edu/cud/about_ud/udprinciples.htm

http://en.wikipedia.org/wiki/Gesture_recognition

Hoda Technology Solderless Touch Screen Kit instruction manual