Advanced Computing: An International Journal (ACIJ), Vol.2, No.6, November 2011

MULTI-TOUCH SCREEN INTERFACES AND GESTURE ANALYSIS: A STUDY

Mrs. Mrudula Nimbarte

Department of Computer Engineering, B. D. C. O. E. Sevagram, Wardha, MS(India)
mrudula_nimbarte@rediffmail.com

ABSTRACT

The way how we handle computers today will soon change. The future technology will allow us to interact with the computer on different level from the current technology what we are used to. The tools such as the mouse and the keyboard need to communicate with the computer will slowly disappear and be replaced with more comfortable and more natural tools for the human being to use. That future is already here. The rate of how touch screen hardware and applications are used is growing rapidly and will break new records in near future. This new technology requires different ways of detecting inputs from the user - inputs which will be made out of on- screen gestures rather than by clicking of buttons or scrolling mouse wheels. In this paper we studied the gestures defined for multi-touch screen interfaces, the methods used to detect them and how they are passed on to other applications.

KEYWORDS

Multi-Touch, Multi- Touch Screen, Multi- Touch Surface, Computing, Gesture Analyzer

1. INTRODUCTION

1.1. Background and Purpose

The goal of this paper is to introduce a gesture analyzing software library, which works as an intermediate layer between the hardware screen and the application(s) projected on it. The application will receive the gesture data as input, along with information on which objects are affected. The options we would expect for objects shown on a touch screen are *select* (multi- and single selection), *rotate*, *scale* and *move* them, so those are the gestures focused on.

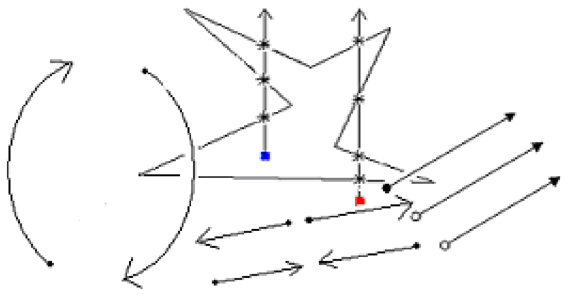

Figure 1.Gesture Symbols

1.1.1 FTIR

FTIR stands for *Frustrated Total Internal Reflection,* which is most widely used representation of the multi-touch technology today. There are occurrences of other methods [1, 6] as well, but FTIR seems to be the method most frequently used to. It is also considered among the cheapest and popular [2] setups.

It is based on very simple principles of the technology:

A pane of a somewhat transparent higher-index medium (acrylic, glass, plastic) is used and Infra Red light is passed into the side of this pane which trapped inside this medium by the refraction index of the material. The infra-red diodes attached to the side of the pane work as light sources in the FTIR-case, while TouchLight[1] uses one infra-red illuminant shining on to the surface.

When the light ray is travelling inside a higher-index medium total Internal Reflection occurs and passes a surface boundary into a lower- index medium. On a finger touch the surface of the pane, causes the light to scatter downwards where it is picked up by an IR camera.

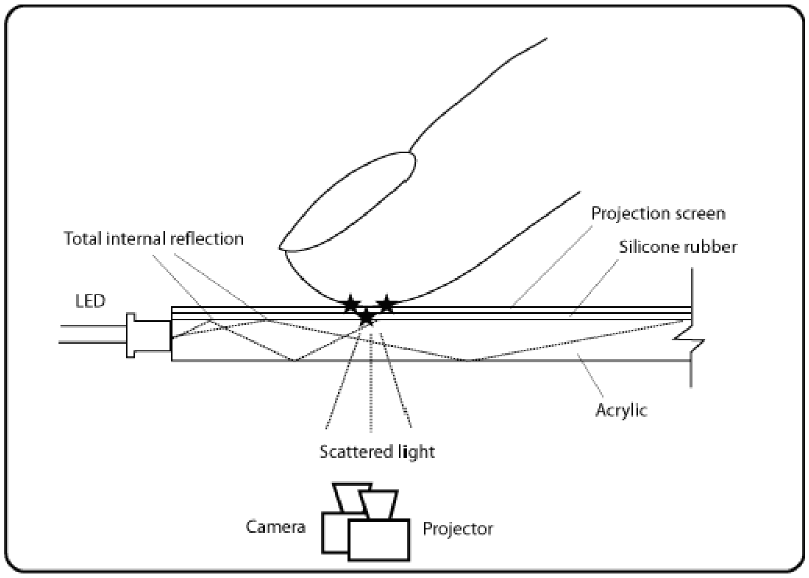

Figure 2.FTIR Technique Overview

A camera on the opposite side from the user with visible light filter registers the scattered light. And a projector projects the image on to the projection screen.

1.1.2 Hardware

The need for instant testing during implementing is always a necessity. Particularly in this case, since the technology developed for is very much considered cutting edge at this time. In many cases unforeseen problems occurred as the project proceeded. Instant testing is decisive in those cases.

The prototype pane was made out of glass material. A glass with attached infra-Red LED's, giving the result as shown in 1.1.1. A back projection material on the opposite side from the user provided the user with a clear image from the projector. We used a web-camera with filter to register the frustrated light to block out the visible light. In our project the filter was made out of a pair of fluorescent-light exposed photo negatives, which were attached to the camera lens. The setup with a little calibration and configuration worked very well for a testing environment.

1.1.3 Multi-Touch

Generally mouse and/or a keyboard are used to interact with a computer. We provide inputs to the computer by the use of buttons. Depending on the input type, the computer can only handle one input at the time which makes the input handling and sorting very easy.

However, one can consider multi-touch is far better from single input handling. The amount of concurrent events in this interface is limited only by the data type holding the number of finger inputs. In the same way the amount of simultaneous users is unlimited, which comes in handy for larger scale display systems. That amount of potential synchronous inputs requires new ways to detect the inputs. Since they are not “on/off” inputs we are used to in the traditional sense, there are needs for new ways to interpret and analyze the input type and the gesture(s) they make out.

The multi-touch screen interface makes the user handling very sensitive. It is in some ways comparable to paper sheets laid upon a table, in the sense that there is not really any need for a manual in how to interact with a piece of paper or move it around. The simplicity of it makes it all very natural for us. Software interaction through a multi- touch interface has the potential of becoming as intuitive and easy for us as moving papers around on a table.

1.1.4 Gestures

In order to achieve the intuitive interactive ways mentioned above, let us define a number of gestures. The ones focused here were as stated earlier; *Move*, *Rotate*, *Select/MultiSelect* and *Scale*. Once these gestures are translated into class objects, the simplicity of interaction will come clear. The cases of *Move* and *Rotate* probably don’t need any introduction, they are self explanatory. Just imagine the gesture movements needed to move or rotate the piece of paper on the table. *MultiSelect* and *Scale* however, are things you don’t generally expect as feasible to a paper sheet.

➢ **Selection/Multi-Selection**

A number of objects are planned onto the screen. The objects within the area of the “line” drawn by the finger(s) are selected, the ones outside are not. To decide what action to take, an event is sent to the application.

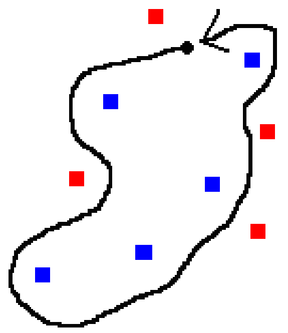

Figure 3.One Finger Selection

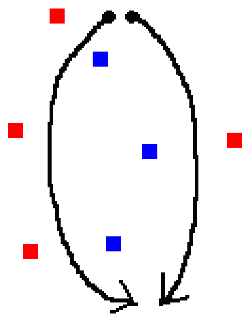

Figure 4.Multi Finger Selection

The red and blue dots are used to represent on-screen objects. The blue ones become selected.

- **Scale**

Scaling requires at least two fingers and is an example of simultaneous inputs. To scale an on screen object, grip it with the fingers and bring them together to scale down, or separate them to scale up.

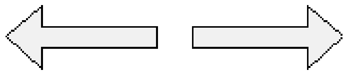

Figure 5.Scale

The scale event is sent to the application that holds the object, with information of the grade of the scaling as well as the object it affects. Quicker finger motion means faster scaling.

1.1.5 External Software

For the finger input detection the gesture library will depend on an external library. The gesture library will collect the input data, then store the data and analyze what kind of gesture it generates. For the purpose of the finger detection it was decided early on to use TouchLib - an open source library for infra red blob detection. The reason for the choice was easy; it filled all the needs for us.

The gesture library will register itself as a listener to TouchLib and thereby get to implement the virtual functions for '*finger down*', '*finger moved*', and '*finger released*'. For each of those events the gesture library will gather 'TouchData', an object consisting of *PositionX*, *PositionY*, *DeltaX*, *DeltaY*, *Area*, *DeltaArea* and a unique *ID*. When a finger is pressed, it receives an ID. The rest of the data gets updated every frame until the finger is released. Of course, the area data has the good consequence of the TouchLib library being pressure sensitive.

TouchLib relies on the use of other libraries as well:

OpenCV Open Computer Vision Library
It is a collection of algorithms and sample code for various computer vision problems.

DSVideoLib DirectShow Video Library
It is a DirectShow wrapper supporting concurrent access to framebuffers from multiple threads.

VideoWrapper Video Camera Libraries
It is a library that provides a single abstract API for interfacing video camera libraries.

OSCpack OSC packets
It is a set of C++ classes for packing and unpacking OSC packets.

2. METHODS

The work was divided into three major parts: Research, Design and Implementation. This is a consistent model which has been used in a number of projects and seems to be the most obvious.

2.1. Research

Since the area of multi-touch still is quite new, there aren't too many well documented projects out there yet. We spent quite some time to get the configuration of our prototype right and to get the TouchLib up and running. Once that was done, we were focusing on getting an understanding of TouchLib, and what we could expect out of it. It comes with some demo applications, so there is some help in getting started.

There is an online open source multi-touch community called the NUI Group, which holds a lot of information on both hardware and software within the FTIR technology.

After a short while we got the dangle of it and could easily write some test applications. The key element is the ID of the finger currently active so each finger is stored into a std::map. The object type holds a std::map of the trail the finger draws, where the object type is of *Vector2* which connects the points for every new position.

It is apparent that the library will have to keep track of the object currently being affected by gestures. Since the application is the only one who has knowledge of its objects, we had to come up with some object mutual to both the application objects and the gesture library. We quickly decided on an idea involving axis aligned bounding boxes. Positions in an on screen application are mutual for everything represented on the screen, and the positions and sizes of the bounding boxes has to be updated by the application and aligned to the objects they represent. Since the number of application windows isn't limited in any way, and since this *is* multi-touch we're talking, there has to be some way to identify the current application too.

2.2. Design

2.2.1 Engine

The goal of the gesture library was for it to be easy to maintain, as well as easily scalable in terms of adding new gestures to it. The applications (i.e. the listeners) should also be separated from TouchLib, so that they just will have to listen to gestures, apart from listening to gesture - and finger events. The gesture library listens to finger events and provides the application with the gestures it detects.

The design went through some changes a couple of times before it finally ended up with the following model:

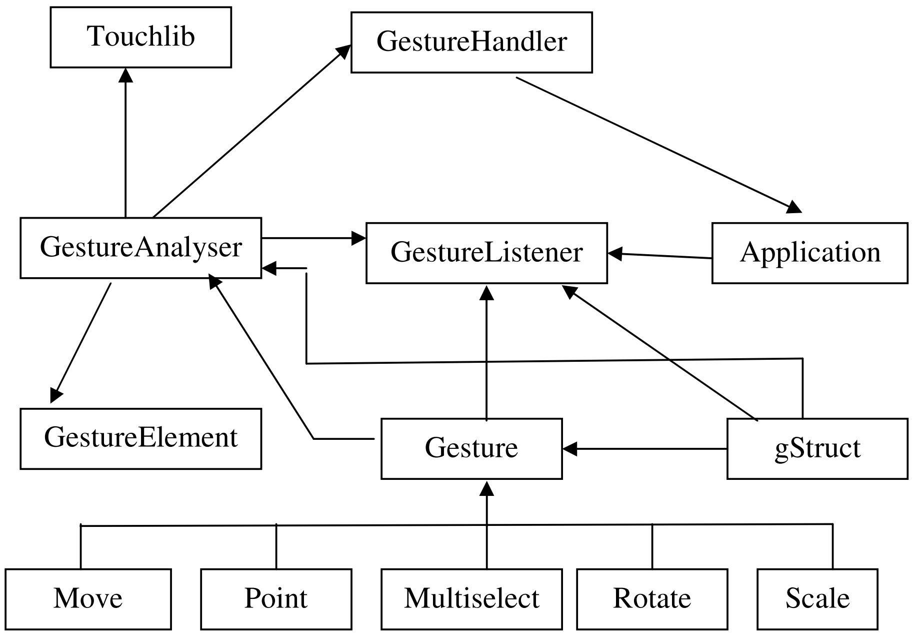

Figure 7.Design Model Overview

The application is separated from the TouchLib and the inheritance structure of the gesture classes provides for the scalability. Every new type of gesture is just added as a new sub class to the *Gesture* base.

The application inherits from the *GestureListener* class and registers itself as a listener at the *GestureHandler*. The application will have to let the gesture library know what objects it will take into account when it comes to analyzing the gestures being performed on them, as well as updated and oriented extreme points of the same objects. One thing all the visible objects in every application have in common is a screen position.

GestureAnalyzer is the class located closest to the hardware. It inherits the TouchLib interface and gets to implement the pure virtual functions that are important for further analysis.*GestureAnalyzer* is a Singleton class, meaning that a new instance of the class is created if one does not exist. Every application registers as a listener in *GestureHandler*, from where an instance of *GestureAnalyzer* is being created. The *GestureHandler* serves as an interface between the application and the TouchLib, separating the two for the reasons mentioned above.

2.2.2 Gestures

The gesture design was more or less a matter of turning hand motions into objects, even though the hand made gestures themselves are hard to define [3]. The interaction should feel as natural as possible, and in no way limited to any pre-defined pattern of how to do this and that. There aren't two people that interact with objects in exactly the same way; in fact, the chance of two gesture movements ever been identical is probably close to zero.

The gesture classes have an inheriting structure, with the base class *Gesture* representing the gesture family. Regardless of what type of gesture that emerges in the analyzing functions of *GestureAnalyzer*, the *Gesture* pointer sent to the listener will have the type of the corresponding sub class.

Point is the simplest to determine of all the gestures. It holds a *TouchData*, which simply is all the data from TouchLib forwarded in one big package. Since it contains all the information of the light blobs' position and size, the listener can decide what to do with it. Single selection of an object is an apparent use of it. Every finger pressed onto the screen will initially compose a gesture of this type.

Rotate requires a minimum of two fingers to perform. The fingers are placed on the screen and moved in a rotating manner.

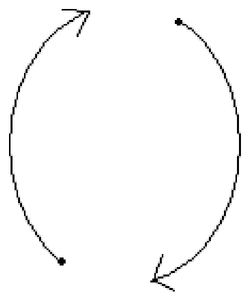

Figure 8.Rotate

The *rotateAngle* variable in the *Gesture* base class will hold the amount of radian rotation from the previous frame. It will have a positive value for clockwise rotations and a negative value during anticlockwise rotations. The listener will get an event of a rotation occurring, with information on the affected *Bbox* along with the updated amount of rotation.

Move is a simple example of a natural interaction with the interface. It is performed by grabbing and dragging the object(s) across the screen. The number of fingers isn't important since the average delta position of the fingers is calculated for as long as the finger(s) are active.

The listener gets the current *Position* - a struct that holds nothing but the x - and y - screen coordinates. These are a part of the *TouchData* object, they are easy to detect.

The *pointOfOrigin* variable gets updated with every new position.

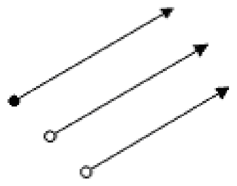

Figure 9.Move

MultiSelect is a gesture that makes it possible to select a number of objects in one stroke. Just as we are used to from traditional interfaces, only that the multi-touch interface invites for new ways to do it. A line is drawn around the objects that are to be selected. One definition of whether an object is inside a given area is the Jordan Curve Theorem. It claims that a particular point is inside a line if, for any ray from this point's position, there is an odd number of crossings of the ray with the line along one of the x - or y - axis.

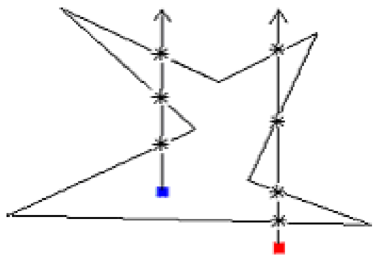

Figure 10.Multiselect

From the blue object to the left in Figure 16, three crossings are found, i.e. the object is inside. From the red object on the right, there are four occurrences of line crossings, which mean that it is outside of the area by definition. The listener will receive a list of *Bbox*'es representing the objects that are affected by the gesture.

Scale is the last of the defined gestures. It is easily detected by analyzing the *DeltaX* and *DeltaY* of the *TouchData*. If the deltas of the finger movement have different signs, they are heading in different directions. If at least two fingers are affecting a *Bbox* in this manner, a *Scale* gesture is being performed on the corresponding object. A *Scale* gesture can as well be treated as a zoom-effect.

The listener receives a multiplying factor, with which the object will be scaled. The *scaleFactor* variable is calculated for as long as the finger is active. If the *scaleFactor* > 0 the fingers are moving apart, else if *scaleFactor* < 0, they are moving together.

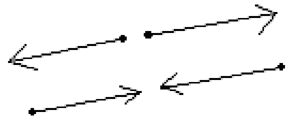

Figure 11.Scale

Note that the *Bbox* representations in every one of the gestures as well might correspond to the main window of the application. This mean that the *Scale* gesture might be treated as a zoom effect and the *Move* gesture can be used as a pan effect.

The gestures themselves aren't predefined in any way. It is up to the listener what to do with them.

2.3. Implementation

The design model described in 2.2 is the final implementation. Here are the pre- conditions for the different gesture events listed.

On *Rotate*

For this gesture, the fingers will have to have different headings, both in the x - and y - direction. The *dX* and *dY* of the *TouchData* may have either a positive or negative value, depending on the current direction.

On *Scale*

To perform this gesture fingers have to be heading in separate directions in either direction.

On *Move*

To perform this gesture every finger involved in the gesture has to be aligned to roughly the same heading.

On *MultiSelect*, analyzing is performed in fingerUp

IsInside performs a bitwise check to see if the first bit of the number of crosses is equal to 1, which is a condition that has to be true for an odd number of line crossings along the axis out of the current *Bbox*.

The gesture of *Point* is performed immediately in the fingerDown function. There isn't really a precondition for it, other than the occurrence of a touch event. The finger will most probably take part in the creation of other gestures during the fingerUpdate function though. As long as a finger is active, it is a part of the objects under observation.

With this model it is necessary to perform a 4 dimensional iteration to locate the correct *Bbox* and provide the listener who owns the belonging object with information on the current gesture event and which object it affects.

3. ADVANTAGES

(i) It is a simple and inexpensive technique. It constructs a multi touch display with the available and less costly materials.

(ii) Scalable technique that enables high-resolution graphics. It provides support to any resolution possible as all multiple points could be generated on a camera.

(iii) Multi touch screen acquires true touch image information at high spatial and temporal resolutions. The actual finger print of the touch is obtained. This could be used to determine the force sensitivity on displays; either too hard or soft touches can be analyzed.

(iv) It is scalable to large installations. Any kind of applications can be made to suit multi touch using FTIR. Allows us to create sophisticated multi-point widgets for applications.

(v) Larger shaped-display systems i.e. it is well suited for use with rear projection like wall screens, table tops. All this lead to high resolution graphics.

4. APPLICATIONS

A multi touch display can be used in

(i) Personal Computers, Laptops, Tabletops, Graphics Tablets.

(ii) It supports both LCD and CRT monitors.

(iii) Telephones, Watches, PDA's, Mobile Phones.

(iv) An advanced multi touch gaming with high graphics support.

(v) Governmental, Offices and business purposes.

(vi) Enhanced multimedia experience including audio, video and photo sharing.

(vii) Enhanced dining experience.

5. CONCLUSIONS

Multi-touch interfaces have the strength and potential to alter the way we work with data and applications, resulting in more dynamic interactions around content. These devices and supporting applications provide diverse ways of visualizing information to improve understanding. They also facilitate new ways to foster collaborative creation, permitting several users to work simultaneously on a single screen.

REFERENCES

[1] Andrew D Wilson, "TouchLight: An Imaging Touch Screen and Display for Gesture-Based Interaction", International Conference on Multimodal Interfaces, October 13–15, 2004, State College, Pennsylvania, USA

[2] Jefferson Y. Han, "Low-Cost Multi-Touch Sensing through Frustrated Total Internal Reflection", Symposium on User Interface Software and Technology, October 23-27, 2005, Seattle, Washington, USA

[3] Vladimir I. Pavlovic, Rajeev Sharma, Thomas S. Huang , "Visual Interpretation of Hand Gestures for Human- Computer Interaction: A Review", Pattern Analysis and Machine Intelligence, IEEE Transactions on, Jul 1997.

[4] Hrvoje Benko, Andrew D. Wilson and Patrick Baudisch, "Precise Selection Techniques for Multi-Touch Screens", Conference on Human Factors in Computing Systems, Proceedings of the SIGCHI conference on Human Factors in computing systems, Montréal, Québec, Canada, 2006.

[5] Dean Rubine, "Specifying Gestures by Example", International Conference on Computer Graphics and Interactive Techniques, Proceedings of the 18th annual conference on Computer graphics and interactive techniques, 1991.

[6] Paul Dietz and Darren Leigh, "DiamondTouch: A Multi-User Touch Technology", Symposium on User Interface Software and Technology, Proceedings of the 14th annual ACM symposium on User interface software and technology, Orlando, Florida, 2001.

Author

Mrs. Mrudula Nimbarte did M.E.(CSE) from PRM Institute of Technology and Research, Badnera and currently working as Assistant Professor since last nine years at Bapurao Deshmukh College of Engineering, Sevagram. Her area of interests is Compilers, System Software, Theory of Computation and Computer Architecture and Organization.