A Unifying Input Framework for Multi-Touch Tables
Michiel Hakvoort
m.c.hakvoort@student.utwente.nl

ABSTRACT
With the uprise of multi-touch table systems, more and more applications which have to deal with multi-touch input, sometimes even multi-user input, are being developed. The development of such applications often introduces the use of frameworks designed to deal with multi-touch input from a specific type of multi-touch table. In cases where no frameworks are available for a specific kind of multi-touch table, it might even be necessary to handle the table's input through a low level API. This, in turn, requires additional layers to be implemented in order to be able to deal with the input in an abstract, usable way.

In this paper an unifying input framework is presented which abstracts away from the hardware limitations of different kinds of touch tables and provides a general interface for handling multi-user, multi-tangible, multi-touch interaction.

Keywords
Multi-touch, multi-user, gestures, DiamondTouch, interaction design, single display groupware, framework, tangible interaction

1. INTRODUCTION
Most user applications developed to date are designed to deal with keyboard and mouse input. Even operating systems heavily rely on either or both input devices in order to deliver the expected user experience. Because of this operating system requirement, operating systems often sustain the development of user applications focused on dealing with mouse and keyboard input. This, in turn, sustains the development of frameworks for dealing with mouse/keybord input where concrete capabilities of the input device are abstracted away so application developers can deal with more complex input actions, like scrolling a mouse wheel, regardless the type of mouse.

With the current growing interest for developing applications for multi-touch hardware, operating system manufactures are falling behind on supporting application developers with a general approach to handle multi-touch input [1]. Because of this, alternatives for multi-touch, sometimes even multi-user, input are being actively developed by the scientific community. These alternatives are often presented in the shape of a framework or toolkit, but in most cases they are only suitable for supporting a specific type of touch table[2, 3, 4]. When a framework does provide support for different types of touch tables, the support is only focused on the common traits between different types of touch table hardware[5], and is thus not generalized into a concept which also encompasses all differences. This means that there is currently no framework available which provides an abstract approach to support the features of all modern multi-touch hardware.

As the availability of good toolkits can support the creativity when creating interfaces [6], application developers should be provided with a framework which enables them to deal with multi-touch input in a logical and flexible way, independent of the type of multi-touch table they are using. Furthermore, extending such a framework for different types of touch table hardware should require as little effort as possible. However, the current unavailability of such a generalized framework raises the question whether it is possible to design and create a unifying framework for dealing with multi-user, multi-touch input from tabletops.

In order to answer this question we will first take a look at related multi-touch input handling frameworks. How are the architectures behind these frameworks set up and how is the input from touch tables presented towards applications developers? Next we will make an assessment of modern multi-touch technologies. The common traits and differences between these technologies are then used to create a general concept on how multi-touch input can be represented. With the details on hardware covered, we examine the required level of abstraction regarding touch input that the framework should provide.

When the details on how the input is received and is represented to the application developer are known, the framework architecture will be discussed. After defining the architecture, the implementation details will quickly be discussed.

Finally, the framework is evaluated through the implementation and testing of a driver for the DiamondTouch multi-touch table[7] and through the implementation and use of an application using the framework. The application will be developed for the DiamondTouch multi-touch table.

2. RELATED WORK
In order to get an impression of what is currently being used for handling multi-user, multi-touch input, we examine a small but varied set of multi-touch input handling frameworks. We do this by examining the layers of the different architectures in order to get an understanding of the variety of approaches to the multi-touch abstraction problem.

2.1 TUIO
Although being a protocol and not a framework, TUIO[8] is a widely used tool in the multi-touch community. The TUIO protocol was designed as a communication interface with the sole purpose of carrying information regarding tangible tabletop controller interfaces over a network. This enables frameworks to distribute information regarding tangible controller interfaces to clients, running on any host and being implemented in any host language, given that both client and framework adhere to the TUIO protocol.

With a variety of TUIO client libraries implemented and available, enabling a client for communication over TUIO the protocol is trivial. Although the TUIO protocol was designed for use with tangible interfaces, its diversity made it possible to be used for multi-touch interaction. TUIO is, however, only designed as a carrier for information regarding tangible interaction, and is thus incapable of carrying information regarding different users on a multi-user, multi-touch table.
2.2 TouchLib

Built for recognizing touches through optical imaging, the TouchLib framework's architecture (see Figure 1) is designed for the analysis of frames from a camera feed. A video camera is pointed towards the surface of a table where, through touching the table, infrared spots are picked up by the camera.

When examining the architecture of the framework we can see that it is closely related to the setup of a video analysis system, as also discussed by François [9]. First there is the preprocessing step, where noise is removed from the input frame and where the frame is converted into a normalized version. This preprocessing step is required to simplify the labeling of the connected components in the segmentation phase. When labeling connected components, adjacent pixels with similar brightness values are considered to be part of the same object. When all connected components are labeled, the bright objects are labeled as blobs. For multi-touch interaction these blobs depict a touch on the surface of the table. After all blobs are extracted from a frame, they are matched against the blobs from the previous frame in the segment tracking step. This matching is done in order to identify the modification, removal or addition of touches on the table surface. Whenever a touch is modified, removed or added all registered listeners for touch events are notified. In addition to the listener model, TouchLib also enables the propagation of touch events through the TUIO protocol.

2.3 reacTIVision

In technical terms closely related to TouchLib is reacTIVision, a framework aimed with the purpose of enabling tangible interaction on a tabletop. Like TouchLib, reacTIVision also recognizes tabletop interaction through the analysis of video input. The main difference between both lies in the steps following the segmentation step (see Figure 2). Where TouchLib handles all recognized blobs as possible touches, reacTIVision makes a distinction between touches by fingers and fiducials. These fiducials are unique visual markers which can be attached to any physical object, allowing for interaction through the physical object on a tablet. After the distinction between fiducials and the touches has been made, touches are being tracked in a way similar to the tracking of touches in the TouchLib framework. As every fiducial is unique, tracking fiducials is not required in order to track the movement of a physical object. Another major difference between reacTIVision and TouchLib is that reacTIVision only provides communication through the TUIO protocol. This means reacTIVision is always to be used as a standalone application, instead of being integrated into an application.

2.4 LibTisch

One of the few frameworks to provide support for different types of touch table hardware is LibTisch. LibTisch is divided into a hardware abstraction layer, a transformation layer, an interpretation layer and finally a widget layer (see Figure 3). The hardware abstraction layer uses a protocol similar to TUIO to propagate touch information from a multi-touch source to the transformation layer. At the transformation layer the touch coordinates are translated into screen coordinates, before passing the touch information to the interpretation layer. At the interpretation layer the framework has the capability of recognizing single-touch gestures within predefined regions. Because of the fact that within a region only single-touch gestures can be recognized, every region is in fact a small single-touch input table. With the link from region to widget, multiple widgets can concurrently receive single-touch input. It is, however, not possible for a single widget to receive multi-touch input.
2.5 DiamondSpin
The DiamondSpin toolkit [4] is a high level toolkit designed to deal with input from the DiamondTouch multi-user, multi-touch table. Unlike the earlier discussed frameworks, the DiamondSpin toolkit does not provide support for multi-touch interaction. Instead it focuses on the DiamondTouch capability to distinct users and provides a multi-user single-touch interaction environment. This can also be seen in the fact that DiamondSpin is closely integrated with the Java Swing widget toolkit, which is a toolkit designed for the interaction through mouse and keyboard. With only one point of touch to track per user, the DiamondSpin toolkit is relatively simple when compared to the frameworks handling multi-touch input (see Figure 4).

3. MULTI-TOUCH TECHNOLOGIES
In order to have an understanding of what properties of different types of touch table hardware should be exposed to application developers, an assessment of the available multi-touch hardware needs to be made. By identifying the common traits and differences between the different types of hardware we will also lay the foundation for a general input format for the framework.

3.1 Optical Recognition Technologies
3.1.1 Frustrated Total Internal Reflection
Adopted by Han in 2005 [10], frustrated total internal reflection is one of the most popular hardware setups for multi-touch tables to date. This technology is based on the physical phenomena of total internal reflection. When a ray of light hits the boundary of a medium it is travelling through at a low enough angle, the ray of light will be reflected instead of being refracted. When passing rays of light through a flat object of a translucent material with a high enough refraction index, like acrylic, the light is essentially trapped inside the material. When a finger touches the translucent material rays of light will hit the surface of the finger and scatter towards the opposite direction of the touch. By placing a video camera below the surface, the scattered light is detected (see Figure 5). By analyzing the frames provided by the video camera, the points of contact can be determined.

3.1.2 Diffuse Illumination
Closely related to an FTIR based touch table, is a diffuse illumination based touch table. Both technologies rely on the input of infrared light. The main difference is found in the fact that the FTIR setup only reflects light whenever an object touches the table surface, due to the frustration of totally internal reflected light. With diffuse illumination, however, all objects in close proximity of the table are recognized. This is because of the fact that infrared light sources are put below the table’s surface and illuminate any object hovering above the surface (see Figure 6). Because any object in close proximity of the table can be recognized, diffuse illumination based touch table can be used to recognize fiducials as well.

3.2 Capacitive Coupling Technologies
A relatively new and complex technology is capacitive coupling. This technology relies on the conductive properties of the human body. In contrast to optical recognition based tables, the sheer complexity of the technology makes capacitive coupling based tables expensive to manufacture. In order to get an impression of the capabilities of capacitive coupling, we will discuss two systems: SmartSkin and the DiamondTouch table.
Because of this, the signals from the antennas are code division multiplexed. A downside of this approach is that it is impossible to detect the exact location of a touch on an antenna. With 129 vertical and 172 horizontal antennas only 301 (129 + 172) points of input are provided, whereas the tables surface can be identified as consisting out of 22,188 (129 × 172) distinct points. This implies that there can be ambiguity on the amount of touches and where the touches are exactly located (see Figure 7). The DiamondTouch is, however, capable of making the distinction between different users through the different receivers, enabling multi-user interaction on the table’s surface.

Figure 7. Five fingertips as perceived by an FTIR based table (left) and the same gesture as perceived by the DiamondTouch table (right)

3.3 Technology Hybridization
A reasonably new development is the hybridization of different technologies with the purpose of augmenting existing multi-touch technologies with new capabilities. For instance the introduction of hand tracking by Dohse et al.[13] enables multi-user interaction on tables which are only capable of recognizing multi-touch input. Also the introduction of tangible interaction through RFID tagged physical objects, as used by the StoryGrid system [14], enables the possibility of augmenting multi-touch technologies with the ability to recognize tangible objects. With technology hybridization emerging we expect more future multi-touch table systems to support tangible interaction and being able to make a distinction between different users.

3.4 Common Traits and Differences
When looking back at the discussed technologies we can state that all interaction occurs on a fixed sized two dimensional surface. Furthermore, the state of all touch input is updated frequently over time. When looking at the differences, we see that some technologies enable the use of tangible objects on the table’s surface. Furthermore, technologies like the DiamondTouch are capable of recognizing touches from different users, but this comes at the cost of introducing ambiguous input.

4. INPUT ABSTRACTION
In order to provide a general solution for different types of multi-touch hardware, an abstract concept for multi-touch input is required. When looking back at the discussed multi-touch technologies we can recognize three different capabilities of multi-touch hardware.

4.1 Multi-Touch
First there is the capability of recognizing multi-touch input. When considering every point at a rectangular two dimensional surface, the point is either being touched or not touched. However, as surfaces do not need to be rectangular shaped, the projection of a non rectangular surface onto a rectangular shape will also contain points which can never be touched. Furthermore, with the concept of ambiguous input introduced by the DiamondTouch table, some types of touch table hardware cannot be certain on whether a point on the surface is being touched. This means that per point on a, possibly projected, rectangular shaped surface we can identify the following states for a point:
- A point is not touched
- A point is touched
- A point can never be touched
- It is uncertain whether a point is touched

Within a single input frame, a two dimensional image of point states can represent all areas being touched on the table’s surface.

4.2 Multi-Tangibility
Second there is the possibility of multi-tangibility. Multiple tangible objects can be placed on a tabletop and each object can be recognized independently. Just like touches, the collection of recognized tangible objects can also be represented in a two dimensional image. In this image each point can refer towards a tangible object. In order to generalize this concept, we will also consider touches by human skin as being tangible objects. The image of tangible objects can then be used in conjunction with the image of point states to identify tangible objects on the table. This way the location of a tangible object can also be marked as being ambiguous.

4.3 Multi-User
Finally we will consider the multi-user capabilities of multi-touch hardware. Just like tangible objects, different users can be represented in a two dimensional image. However, when using the user image in conjunction with the tangible image, we see that it is possible for a tangible object to be associated with multiple users. As touches by human skin are also considered as being tangible objects, a touch from one human could theoretically originate from different users. Because of this, we consider every tangible object to have at most one user. This also means that there is no longer the need of representing the different users within a two dimensional image. This way, touches from different users will have to be represented by different tangible objects.

4.4 Abstraction Model
With these abstractions defined we can define a single input frame as consisting out of a two dimensional image of point states and a two dimensional image of tangible objects. With every tangible object being associated with at most one user, we also introduced the abstraction of the multi-user capability. However, as a single frame can contain ambiguous data, it is possible for point states from multiple users to overlap. By using a collection of frames instead of a single frame to represent input, this problem can be overcome. This leads us to the general input abstraction model (see Figure 8) which is to be used by the framework, providing support for multi-touch, multi-user and/or multi-tangibility enabled tabletops whilst recognizing the possible presence of ambiguity.

Figure 8. The multi-touch, multi-user, multi-tangibility abstraction model
5. TOUCH ABSTRACTION
For making the framework usable for application developers an abstract concept of touch interaction is required. This abstraction should at least encompass all information contained by the general input format. The other properties of the abstract concept of touch interaction are discussed separately in this section.

5.1 Touch Tracking
First we consider that the act of touching a tabletop surface does not occur at a single moment. Instead, a touch begins at the moment the table is first touched and lasts for as long as the surface is being touched. When touching a surface the location at which the surface is touched can also shift over time. By tracking this motion a touch’s lifetime is extended until it is released and the act of touching will no longer be a single moment event but a lasting event. The tracking of a touch can be beneficial for the recognition of complex gestures, which incorporate movement [15].

5.2 Touch Shape
The area of contact for a touch with the table denotes the shape of a touch. As the shape of a touch can be of great importance for recognizing still gestures [16], the shape of a touch should be exposed towards application developers.

5.3 Touch Joining and Splitting
As the input from a tabletop lies within a two dimensional plane, a touch cannot be occluded by any other touch. Because occlusion is not possible, disappearance of a touch implies that the touch has either been released or that the touch was joined with another touch. For the appearance of a touch, a similar concept applies. Either a new touch is touching the table’s surface, or a touch has been split into several new touches (see Figure 9). The advantages of this approach is that it enables the recognition of a wide range of new gestures. In combination with still gesture recognition it is for instance possible to recognize the positioning of a hand on the table surface (see Figure 10).

6. FRAMEWORK ARCHITECTURE
In order to present the earlier discussed abstract concept of multi-touch interaction towards application developers, the framework’s architecture needs to be designed to be capable of transforming the abstract input format into the concept of multi-touch interaction. Because the framework only accepts the discussed input format, raw input from different types of multi-touch hardware needs to be preprocessed first. Also, the framework only focuses on enabling the development of multi-touch applications using the discussed concept of touch abstraction. This means that gesture recognition or widget integration is not within the focus of the framework’s architecture (see Figure 11). The steps taken to process the input frames are discussed in the remaining part of this section.

6.1 Segmentation
After reading a collection of frames from an input source, each frame is segmented. In this segmentation step areas of adjacent points containing identical tangible objects are identified and labeled as segments. These segments are marked as being either ambiguous or non-ambiguous depending on whether or not the segments contains a point state which is marked as being ambiguous. When all segments are collected from the frames, these are passed to the tracking layer.

6.2 Segment Tracking
After all segments are extracted from the frames, the segments are matched against the collection of tracked touches. This matching process searches for a best matching touch for each segment. Because a touch is allowed to be picked as a best match for multiple segments, multiple segments may be identified with a single touch. In order to allow as many touches as possible to be tracked, every touch is also matched against the collection of segments. As a consequence of this, multiple touches may also be identified with a single segment. With every touch being identified with zero, one or more segments and every segment being identified with zero, one or more touches, these relations are passed to the trajectory analysis layer.
6.3 Trajectory Analysis
In the trajectory analysis layer new touches are identified and existing touches are marked as either being tracked, split, moved or released (see Figure 12). As the splitting or joining of touches results in the creation of new touches, trajectory analysis layer also marks the new touches as being newly identified.

6.4 Event Generation
With the trajectories analyzed, events will be generated and fired. Whenever a new touch is identified on a multi-touch table, the multi-touch table’s listeners are notified with the registration of the new touch. However, after this notification, new events regarding the registered touch will no longer originate from the multi-touch table but from the touch itself. In order to receive events from a touch, a listener can be set to listen to events from the touch. By generating events per touch application developers can focus on the actions of a single touch or on the actions of a collection of touches, instead of the actions of the multi-touch system as a whole.

7. IMPLEMENTATION
Based on the discussed theoretical background, an implementation of the framework has been written for the Java platform. The choice for the Java platform lies in its platform independency and the availability of libraries for communicating with the DiamondTouch table, enabling the implementation of a DiamondTouch driver for the framework. In the remainder of this section we will briefly discuss the software design of the framework, the implementation of the segmentation layer, the segment tracking layer and the possibility of disambiguating input from multi-touch table systems that can provide ambiguous input.

7.1 Software Design
The simplified class diagram of the framework (see Figure 13) provides as an introduction to some of the concepts behind the design of the framework. First there is the TouchStream which is responsible for preprocessing the raw input into frames. When new frames are available, a TouchDevice requests the TouchStream for the new frames to be copied into the FrameBuffer. As these frames can be several megabytes in size the use of a FrameBuffer ensures that the framework will require a near constant amount of memory during operation. After the new frames are copied into the FrameBuffer, the frames are being processed by the TouchDevice. The TouchDevice incorporates the segmentation, tracking, trajectory analysis and event generations layers of the framework. Listeners of the TouchDevice are notified when a new Touch is registered. Any other notifications regarding Touch related events are fired from a Touch object itself.

7.2 Segmentation
For extracting all segments from a single frame, all points within the areas of adjacent points containing identical tangible objects are grouped. The grouping of points is done in linear time through an algorithm which gathers all adjacent points of one tangible object at a time. After all points are visited and all tangible objects are grouped into segments, the marching squares algorithm is used to identify the concave hull of these segments. This concave hull is then simplified through applying a polygon simplification algorithm. When the segment is promoted to a touch, the simplified concave hull is exposed to application developers as the shape of the touch.

7.3 Segment Tracking
For matching a touch with a segment or a segment with a touch, every touch-segment pair is given a measure of confidence. When the measure of confidence exceeds a predefined threshold the identification of a segment with a touch, and vice versa, is plausible. The confidence measurement used in the framework is inspired by the confidence measurement used by François [9], meaning it takes the size of a segment and the projected location into account. The main difference between the new approach and the approach used by François is that the penalty on the difference in area size of a segment and area size of a touch is also dependent on the size of the table’s surface. Small segments and touches will receive a lower penalty when the relative difference between the area sizes of both is large than larger segments and touches will receive.

7.4 Touch Disambiguation
Earlier we have discussed the possibility of ambiguity of touches caused by multi-touch table hardware, for example by the DiamondTouch table. Although it is impossible to disambiguate the touches within a single frame, when assuming that at most one point of touch is added to the surface at a time, successive frames can provide useful for partially solving the ambiguity problem. Consider a single point of touch is already active and being tracked on the DiamondTouch table. When another point of touch is added by the same user, but at a
different vertical and horizontal antenna, the resulting image contains four points of touch. However, under the assumption that at most one point of touch can be added at a time we know that from the three newly detected points of touch, two must be the side effects of ambiguity. Identifying which two touches are side effects is in this case trivial, as we know that the side effects must share an antenna with the point of touch already being tracked. As successive frames will also contain the side effects, not keeping track of the side effects will result in the delayed and false recognition of the side effects as being points of touch. By promoting the side effects to normal touches with the distinction of not being able to generate any events, which includes join and split events, the tracking of side effects can directly be combined with the tracking of touches. Unfortunately, when a normal point of touch is split it is not possible to detect a side effect. This implies that when points of touch cross on an antenna, false touches are still being generated.

8. DISCUSSION
With the research completed and the framework implemented, the framework is tested through the development of a driver for the DiamondTouch and an the development of an application using multi-user, multi-touch capabilities.

8.1 DiamondTouch Driver

8.1.1 Implementation
In order to test the ability of the framework to be extended with new types of multi-touch hardware, a driver for the DiamondTouch table is written. This driver, an implementation of the TouchStream class, uses the Java library provided by MERL to communicate with the DiamondTouch table, which resulted in a fully functional framework driver of just under 200 lines of code. As the DiamondTouch can provide ambiguous input information, the driver returns multiple frames containing touch data per input handling cycle. In order to create a single frame for a single user, every active vertical antenna for that user is combined with every active horizontal antenna. By combining all of these locations an image of ambiguous point states is created. Next to this, the tangible object image is used to store a tangible object carrying information regarding the specific user.

8.1.2 Testing
The driver was tested by a quickly implemented virtual painting application. This painting application would assign a random color to each new touch and whenever the touch is moved, the virtual paint is applied on the point of touch. When testing for disambiguation not a single side effect was filtered by the framework. A closer study to the cause of the failing disambiguation revealed that the DiamondTouch does not synchronize the signals that run through the horizontal antennas with the signals that run through the vertical antennas. This problem can be overcome by delaying the recognition of side effects and points of touch by a single input handling cycle. When side effects occur right after a point of touch is recognized, the disambiguation method can be applied.

8.2 FeelSound
The FeelSound application is an application which synthesizes real-time sounds when touching the DiamondTouch surface. The earliest version used the table’s horizontal axis for denoting the frequency of the sound and the vertical axis for denoting the amplitude of the sound. When multiple users and/or fingers were touching the table’s surface the different synthesis would be mixed. Next to the emission of synthesized sound, FeelSound also returned visual feedback by displaying the points of touch.

8.2.1 Implementation
The implementation of FeelSound was comparable to the implementation of the virtual painting application. By using a synthesizer library, registering a new synthesized sound and modifying its properties real-time was achieved in a couple lines of code.

8.3 Other Considerations
As the framework itself is not capable of recognizing fiducials by itself, writing a preprocessing layer for a hardware setup that supports fiducials result in the need of writing a segmentation layer in the preprocessing layer. With segmentation occurring twice, supporting fiducials comes at the cost of performance loss. Other types of touch table hardware which also require special processing steps can suffer from the same drawback. However, when enabling the framework for input from the TUIO protocol, it is able to recognize fiducials right away. This can still come at the cost of performance loss, but the load can be shared over different machines.

9. CONCLUSIONS AND FUTURE WORK
In this paper we have presented a framework capable of unifying the common traits and differences of modern multi-touch table hardware. Next to this, the abstraction of multi-touch interaction provided by the framework gives application developers an accessible way to enable the recognition of a range of new gestures. The framework’s architecture also proved to be suitable for the disambiguation of ambiguous input from a multi-touch table like the DiamondTouch. Furthermore, just like extending the framework to work with the DiamondTouch, it can be extended for other types of multi-touch table hardware too by writing relatively simple drivers which preprocess the hardware input. Is it possible to create a unifying framework for dealing with multi-user, multi-touch input from table tops? The answer to this question is yes. However, as considered in the discussion, the unification process may come at the price of performance loss.

As the framework is solely responsible for the tracking of segments, future work can be focused on the improvement of segment tracking techniques for multi-touch interaction. This room for improvement also holds for the trajectory analysis phase which is currently only capable of joining and splitting similar sized touches. Furthermore, in order to allow more people to adopt the framework both a TUIO input and a TUIO output driver can be written. Next to this, the framework can be a good basis for use with optical based multi-touch tables and a good platform for researching multi-touch technology hybridization and multi-touch interaction.

To conclude, by releasing the framework under an open source license, I hope to provide the academic community with a good toolkit which sparks interface creativity.

REFERENCES


