Multi-Touch Gesture Recognition using Feature Extraction

Francisco R. Ortega, Naphtali Rishe
School of Computing and Information Sciences
Florida International University
Miami, FL. USA
Forte007@fiu.edu, NDR@acm.org

Armando Barreto, Fatemeh Abyarjoo, Malek Adjouadi
Electrical and Computer Engineering Department
Florida International University
Miami, FL. USA
BarretoA@fiu.edu, Fabya001@fiu.edu, Adjouadi@fiu.edu

***Abstract*—We are motivated to find a multi-touch gesture detection algorithm that is efficient, easy to implement, and prospectively scalable to real-time applications using 3D environments. Our approach tries to solve the recognition for gestures with the use of feature extraction without the need of any previous learning samples. Before showing our proposed solution, we describe some algorithms that attempt to solve similar problems. Finally, we describe our code to accomplish off-line gesture recognition.**

I. Introduction

We present the initial development of our approach to detected gestures in a multi-touch display using a finite state machine and feature extraction. We described our methods in the context of important previous work, like the $1 algorithm [1] and the Rubine algorithm [2].Our first approach to the problem is demonstrated with off-line data.

Gesture detection algorithms are not a new problem in Human-Computer Interaction (e.g., [3], [4]) and some have derived from stroke detection algorithms [5],as will be shown in our brief review of previous work. With the availability of multi-touch devices such as the iPad, iPhone and desktop multi-touch monitors (e.g., 3M M2256PW 22" Multi-Touch Monitor) new concepts have developed in order to help the transition to a post-Windows-Icon-Menu-Pointer (WIMP) era. The development of Natural User Interfaces (NUIs) presents many exciting challenges.

To solve the problem, we first reviewed previous work in the area of stroke and gesture detection relevant to our solution. After the related work section, we cover our proposed solution, and future direction.

II. Background

There have been various approaches to gesture recognition, like the use of finite state machines [6], [7], Hidden Markov Models [8], neural networks [9], dynamic programming [10], featured-based classifiers [2], and template matching [1], [11]–[13]. Thorough reviews can be found in [14]–[16]. For our work, we have looked in-depth at feature extraction recognition methods like the Rubine and $1 algorithms [1], [2].

Some of the methods used in handwriting recognition [15] can be used for gesture recognition [17]. While handwriting recognition efforts date as far back as the 1950s [15], it has been the work of Rubine [2] that has been used as a foundation by some in the gesture recognition area [1], [18] including us. While the Rubine algorithm [2] uses training with its features, we aim at finding features that can be used without training samples.

We called the "$ algorithms" a partial list of approaches derived or inspired by work from $1 algorithm [1] such as [12], [13], [19], [20]. The $1 algorithm [1] provides a simple way to develop a basic gesture detection method. In contrast, algorithms based on Hidden Markov Models or neural networks [9] involve a high level of complexity for the developer and the system as well. $1 provides a very fast solution to interactive gesture recognition with less than 100 lines of code [1] and requires a simple training set. However, this is not meant for the recognition of multi-touch gestures and for prospective real-time gesture recognition. This does not diminish in any way the importance of the $1 algorithm because there are several features that make it important. For example, the obvious resampling of the gesture, the indicative angle ("the angle formed between the centroid of the gesture and [the] gesture's first point" [1]), and the re-scaling and translation to a reference point to keep the centroid at (0,0). Another important contribution is the use of the Golden Section Search [21] to find the right gesture. The "$ Algorithms" provide a rich set of contributions for the multi-touch research community.

The $N algorithm [12], with the double amount of code (240 lines), improves the $1 algorithm [1] to allow single strokes and rotation invariance discrimination.For example, to make a distinction between A and $\forall$, rotation must be bounded by less than $\pm 90^\circ$ [12]. The $N algorithm [12] was extended primarily to allow single strokes to be recognized. This algorithm also supports automatic recognition between 1D and 2D gestures by using "the ratio of the sides of a gestured's oriented bounding box (MIN-SIDE vs MAX-SIDE)" [12]. In addition, to better optimize the code, it only recognizes a sub-set of the templates to process. This is done by determining if the start directions are similar, by computing the angle formed from the start point through the eighth point. A common feature of the $1 and $N Algorithms [1], [12] is the utilization of the Golden Section Search [21].

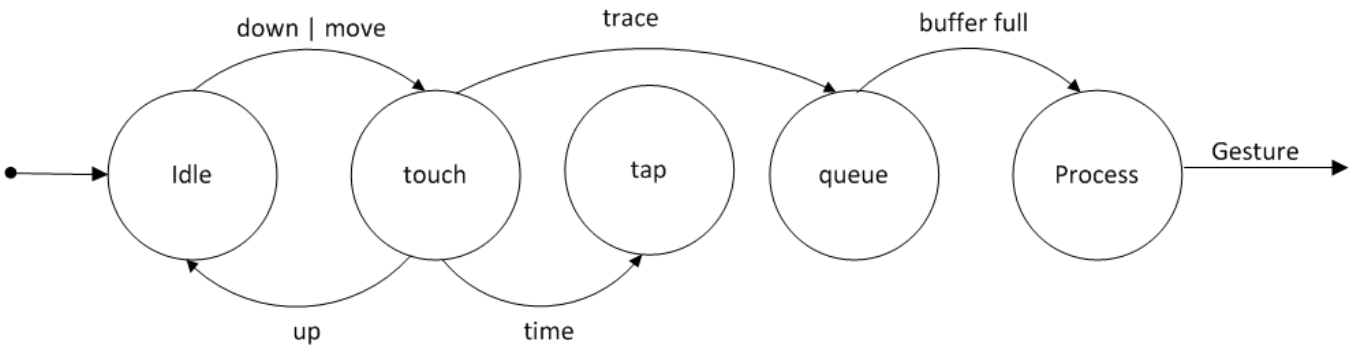

Figure 1: State Machine

Additional algorithms provide great resources for future work. Dean Rubine provides an excellent set of features to be tested with multi-touch data. In addition to the Rubine algorithm [2], we can use Wang et al. [22] to find if the gesture was created with fingers in oblique position or not. Additional information can be found in our previous work [23]

III. Proposed Solution

A. Motivation

We are motivated to find a low-complexity implementation and a fast algorithm that can be utilized for high-demanding applications. In our case, our end goal is to use it with high-demanding 3D navigation environments using multi-touch displays. First, we are motivated by the work of Rubine [2] to find the best features to characterize a gesture while keeping the complexity low, as done by the $1 algorithm [1]. Second, we are motivated to find an unsupervised method of recognition that can be used in high-demanding real-time applications such as 3D synthetic worlds. Finally, we came across a multi-touch gesture detection problem statement created by Greg Hamerly in the ICPC 2012 Competition[1] [24] which resembles some of the work we have been doing with gesture detection. It is the combination of all the great work already mentioned plus our application needs, that have given us the path to follow in this proposed solution. Therefore, we believe that the path we propose, could serve as another building block for the development of POST-WIMP era interfaces.

B. Setup

We used Windows 7 multi-touch technology [25] with Microsoft Visual Studio and a 3M M2256PW Multi-Touch Monitor to test our work and we have tested our approach with off-line data. Windows 7 provides either pre-defined touches or raw touches when using their technology. We chose to work with raw touches because it gives us the flexibility to create custom gestures and to test different methods for detection.

When using raw touches, most systems where multi-touch is available (e.g., iOS, Windows) will provide a "trace" which contains a set of points with coordinates x and y as well as a timestamp for each point. The system will also generate events when the trace is created, moved, and finished. Each touch has a unique identification (ID) that is given at the moment of TOUCHDOWN, to be used during the TOUCHMOVE, and to end when TOUCHUP has been activated. The ID gives us a way to group points from each finger. For specific information on how Windows 7 handles multi-touch technology, please see [25].

C. Method

We begin by combining feature extraction with a finite state machine [26], as shown in Figure 1. The idea is to allow a state machine to keep control of the process in order to find a common system to make the changes. The state starts as idle like for any other input device. Here we have a state transition to the touch state with either down or move events. Once the finger has been lifted, we have a state transition back to idle with the event up. Once the touch state has been reached, the decision must be made to identify it as either a tap (e.g., double tap, 2 finger tap) or a trace. (Note that the user will either be creating traces for a given gesture or a tap. It is our choice not to consider the tap as a gesture). If it is a trace, a transition follows to add this trace data to a thread-safe queue [27]. The queue is used to keep storing the traces while a specific window of data is processed. We can think of this window as a buffer. Once the queue is full, a transition will take place to the "process" state, where Algorithm 1 takes over.

Figure 2 gives more details about the queue and how we are using it. In the left side of the figure, we can see a user creating a three-finger gesture. One option for the user is to create a desired gesture and lift the hand. If this was the only case, then one can just process trace data as they are coming in with a small buffer. However, this brings us to the second case. Here the user can perform multiple gestures

[1]Francisco Ortega leads one of the programming competition tutoring sessions at Florida International University.

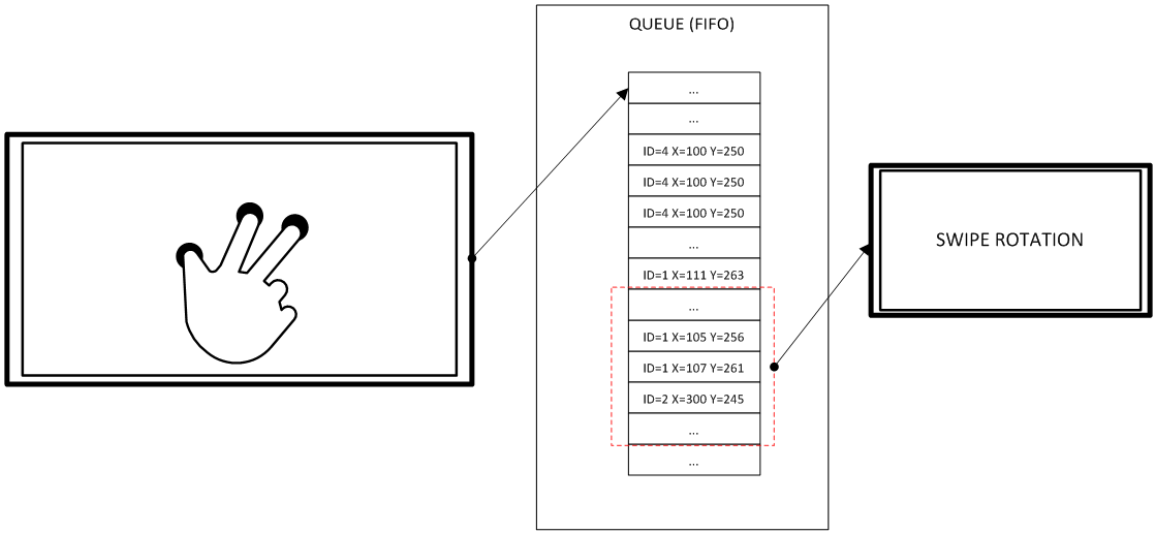

Figure 2: Queue

while leaving the hand in the screen. This is the reason that we need to have a window size long enough to detect a gesture while keeping the rest in the queue. Figure 2 shows this thread-safe queue to have a window size, for a given gesture.

D. Algorithm

Algorithm 1 detects swipe (translation), rotate ,and pinch in/out (zoom in/out). If a system does not provide traces, one of the many clustering techniques available can be used to create them [28]. Because of the importance of a fast detection, any necessary pre-computations must be performed while the traces are added to the queue. This is why, when running the algorithm, is it expected to have the grip computed.

Our primary motivation is to lower the running time of the gesture detection in order to use it with demanding 3D applications. Therefore, the running time of the algorithm is as important as the complete utilization of all the resources available in the system. In this context, it is important to note that the gesture detection runs in its own thread. For more details about this multi-threaded approach, a C++ implementation can be found in [27] or a more detailed explanation with Java code can be found in The Art of Multiprocessor Programming [29].

Algorithm 1 starts by popping the buffer window in line 1. Because there are no clear initial and final snapshots, we assume that by popping the first half of the buffer and popping the rest of the buffer we can obtain an initial and final state of the traces. Because this was tested with off-line data, the data was already created with initial and final snapshots. Lines 2 and 3 assign each snapshot. Before we continue, we must define the variables grip, spread, trace, trace vector, angle rotation and traces.

Traces is a set that contains information for the path taken by each finger. In other words, for each finger, a set of properties is pre-computed, which is called **trace** in the algorithm. For example, the x and y coordinates is the average of a given trace, as shown in Equations 1 and 2 (for the y coordinate, replace the x for the y) already calculated per trace. Note that the variable n in the formulas refer to the total x,y points for a given trace. Because we are dividing the buffer into two snapshots, each snapshot has its own average. A **grip** is defined by the average of all points in each snapshot, as shown in Equation 3. A **trace vector** is defined as trace minus the grip, as shown in Algorithm 1 lines 12 through 15. The **spread** is given by lines 18-19 in Algorithm 1, which calculate the spread as the average difference between the grip point and the touch vector. Finally, the **angle rotation** is the average of the angle obtained by atan2 [30]. In other words, this is the angle between the final touch vector and the initial touch vector.

Finally, the chosen gesture is given by any of the three distance variables (swipeDistance, rotDistance, or zoomDistance)) with the highest values found in Algorithm 1. The swipe distance is given by the spread of the first trace and the grip. The rotate distance is given by the arc length. This is the average angle obtained in line 20 and the radius of the swipe distance. Remember that atan2 [30] values range between $\pm\pi$. In order to obtain the distance, the proper factor 2 must be multiplied as shown in Equation 4 The zoom distance is given by the average final spread distance and the average initial spread distance.

Once everything is computed in the for loop, all we have left to do is to determine the correct gesture. The gesture detected is assigned according to the highest distance value of the swipe distance, rotation distance or zoom distance.

Additional information can be obtained for specific detected gestures. For example, if the gesture detected is a zoom gesture, then additional information can be found, such as if the direction of the user is inward or outward. While the primary goal of the algorithm is to find the correct gesture, additional information is important to be precise about the gesture. Algorithm 1 concentrates in finding the gesture type.

Algorithm 1 GestureDetection

Require: TouchCount ¿ 0

$traces \leftarrow traceQueue.getWindow()$
$iTrace \leftarrow traces.getHalf()$
$fTrace \leftarrow traces.getHalf()$
$iGrip.x \leftarrow iTrace.getGrip.x$
$iGrip.y \leftarrow iTrace.getGrip.y$
$fGrip.x \leftarrow fTrace.getGrip.x$
$fGrip.y \leftarrow fTrace.getGrip.y$
$ivFirst.x \leftarrow iTrace[1].x - iGrip.x$
$ivFirst.y \leftarrow iTrace[1].y - iGrip.y$
$swipeDistance \leftarrow sqrt(ivFirst.x^2 + ivFirst.y^2)$
for $t = 1$ to $traces.Count$ **do**
  $iv.x \leftarrow iTrace[t].x - iGrip.x$
  $iv.y \leftarrow iTrace[t].y - iGrip.y$
  $fv.x \leftarrow fTrace[t].x - fGrip.x$
  $fv.y \leftarrow fTrace[t].y - fGrip.y$
  $di \leftarrow sqrt(iv.x^2 + iv.y^2)$
  $df \leftarrow sqrt(fv.x^2 + fv.y^2)$
  $iSpread \leftarrow iSpread + di$
  $fSpread \leftarrow fSpread + df$
  $angle \leftarrow atan2(fv.y - iv.y, fv.x - iv.x)$
  $rotAngle \leftarrow rotAngle + angle$
end for
$iSpread \leftarrow iSpread/traces.Count$
$fSpread \leftarrow fSpread/traces.Count$
$rotAngle \leftarrow rotAngle/traces.Count$
$zoomDistance \leftarrow fSpread - iSpread$
$rotDistance \leftarrow rotAngle/360.0 * 2 * \pi * swipeDistance$
return Gesture With Highest Distance

$$iTrace[id].x = \frac{1}{n/2} \sum_{i=0}^{\frac{n}{2}-1} trace[id][i].x \tag{1}$$

$$fTrace[id].x = \frac{1}{n/2} \sum_{i=\frac{n}{2}}^{n-1} trace[id][i].x \tag{2}$$

$$iGrip.x = \frac{1}{n/2} \sum_{t \in \mathbf{iTrace}} iTrace[t].x \tag{3}$$

$$rotDistance = \frac{\Theta}{360} 2\pi r \tag{4}$$

IV. Future Work

We believe that the combined approach of feature extraction and state machine for gesture detection provides a fast and easy-to-implement solution. In the quest of searching for different solutions, one can only hope to reach such elegant solution as the seminal work by Buxton [31]. Can we find a one-model-fits-all approach? This is a question that we hope to answer in the near future.

A follow up question that we would like to address in our future work is: Can we detect all possible multi-touch gestures with a combination of feature extraction and finite state machine in real time? We believe that this can be done. Our next phase will be to expand our work and test it in real time.

V. Conclusion

Multi-touch displays have become more widely used and are a standard of one of the de-facto NUI devices that will shape the post-WIMP era. In this paper, we have outlined some valuable previous contributions to the area of stroke and gesture recognition. This review of the literature provides a context for our approach and the reasoning behind our algorithm.

We proposed an algorithm and a set of concepts to allow gesture detection while using off-line data for multi-touch displays. This paper defined concepts such as grip, touch vector, trace and traces to allow the understanding of our algorithm. We explained how to implement Algorithm 1 with the emphasis on an efficient and easy to implement approach.

The next step in the development of our approach is to evaluate its efficiency using real-time data while adding more gestures. As already expressed, our end goal is to use fast gesture detection algorithms for high-demanding applications running in 3D environments.

Acknowledgments

This work was sponsored by NSF grants HRD-0833093, and CNS-0959985. Mr. Francisco Ortega is the recipient of a GAANN fellowship, from the US Department of Education, at Florida International University.

References

[1] J. Wobbrock and A. Wilson, "Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes," *Proceedings of the 20th annual ACM symposium on User interface software and technology (UIST '07)*, 2007.

[2] D. Rubine, "Specifying gestures by example," *ACM SIGGRAPH Computer Graphics*, vol. 25, no. 4, pp. 329–337, 1991.

[3] G. Nielson and D. Olsen Jr, "Direct manipulation techniques for 3D objects using 2D locator devices," *Proceedings of the 1986 workshop on Interactive 3D graphics*, pp. 175–182, 1987.

[4] M. Chen, S. Mountford, and A. Sellen, "A study in interactive 3-D rotation using 2-D control devices," *ACM SIGGRAPH Computer Graphics*, vol. 22, no. 4, p. 129, 1988.

[5] H. Lü and Y. Li, "Gesture avatar: a technique for operating mobile user interfaces using gestures," in *CHI '11: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM Request Permissions, May 2011.

[6] P. Hong and T. Huang, "Constructing finite state machines for fast gesture recognition," *15th International Conference on Pattern Recognition (ICPR'00)*, vol. 3, p. 3695, 2000.

[7] P. Hong, T. Huang, and M. Turk, "Gesture modeling and recognition using finite state machines," *IEEE Conference on Face and Gesture Recognition*, Mar. 2000.

[8] T. Sezgin and R. Davis, "HMM-based efficient sketch recognition," *Proceedings of the 10th international conference on Intelligent user interfaces (IUI '05)*, 2005.

[9] J. Pittman, "Recognizing handwritten text," in *Human factors in computing systems: Reaching through technology (CHI '91)*, New York,NY, 1991, pp. 271–275.

[10] S. MacLean and G. Labahn, "Elastic matching in linear time and constant space," *International Workshop on Document Analysis Systems 2010 (DAS '10)*, 2010.

[11] L. Kara and T. Stahovich, "An image-based, trainable symbol recognizer for hand-drawn sketches," *Computers & Graphics*, vol. 29, no. 4, pp. 501–517, 2005.

[12] L. Anthony and J. Wobbrock, "A lightweight multistroke recognizer for user interface prototypes," in *Proceedings of Graphics Interface 2010 (GI'10)*, Toronto, ON, 2010.

[13] Y. Li, "Protractor: a fast and accurate gesture recognizer," in *Proceedings of the 28th international conference on Human factors in computing systems (CHI '10)*, New York, NY, 2010.

[14] G. Johnson, M. Gross, and J. Hong, "Computational support for sketching in design: a review," *Foundations and Trends in Human-Computer Interaction 2*, 2009.

[15] C. Tappert and C. Suen, "The state of the art in online handwriting recognition," *Pattern Analysis and ...*, 1990.

[16] R. Plamondon and S. N. Srihari, "Online and off-line handwriting recognition: a comprehensive survey," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 1, pp. 63–84, 2000.

[17] M. Blumenstein, B. Verma, and H. Basli, "A novel feature extraction technique for the recognition of segmented handwritten characters," in *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on*, 2003, pp. 137–141.

[18] B. Signer, U. Kurmann, and M. C. Norrie, "iGesture: A General Gesture Recognition Framework," in *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, 2007, pp. 954–958.

[19] S. Kratz, "Protractor3D," in *Proceedings of the 15th international conference ...*, 2011.

[20] ——, "A $3 gesture recognizer: simple gesture recognition for devices equipped with 3D acceleration sensors," in *Proceedings of the 15th international conference ...*, 2010.

[21] W. H. PRESS, B. P. FLANNERY, S. A. TEUKOLSKY, and W. T. VETTERLING, *Numerical Recipes*, 3rd ed., ser. the art of scientific computing. Hong Kong: Cambridge University Press, 2007.

[22] F. Wang, X. Cao, X. Ren, and P. Irani, "Detecting and leveraging finger orientation for interaction with direct-touch surfaces," *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, pp. 23–32, 2009.

[23] F. R. Ortega, A. Barreto, N. Rishe, and M. Adjouadi, "Towards 3D Data Environments using Multi-Touch Screens," pp. 118–121, 2012.

[24] (2012, 11). [Online]. Available: http://cs.baylor.edu/~hamerly/icpc/qualifier_2012/

[25] Y. Kiriaty, L. Moroney, S. Goldshtein, and A. Fliess, *Introducing Windows 7 for Developers*. Microsoft Pr, Sep. 2009.

[26] M. Sipser, *Introduction to Theory of Computation*, 2nd ed. Cengage, 2006.

[27] A. Williams, *C++ Concurrency in Action: Practical Multithreading*, 1st ed. Manning Publications, Feb. 2012.

[28] G. Gan, C. Ma, and J. Wu, *Data Clustering: Theory, Algorithms, and Applications (ASA-SIAM Series on Statistics and Applied Probability)*. SIAM, Society for Industrial and Applied Mathematics, May 2007.

[29] M. Herlihy and N. Shavit, *The Art of Multiprocessor Programming*, 1st ed. Morgan Kaufmann, Mar. 2008.

[30] F. Dunn and I. Parberry, *3D Math Primer for Graphics and Game Development, 2nd Edition*, 2nd ed. A K Peters/CRC Press, Nov. 2011.

[31] W. Buxton, "A three-state model of graphical input," *Human-computer interaction-INTERACT*, vol. 90, pp. 449–456, 1990.