A $1.5 Gesture Recognizer as Input Device for a Multi-Platform Virtual Game Controller

Andreas Schmidt · Ulrich Schwanecke · Friedrich Volmering · Marc Kamradt

Abstract Current high-end computer games mostly are developed simultaneously for several different platforms like e.g. PC, XBox© or Playstadion©. These systems often have varying individual input devices. The necessity to consider all of them during the development process potentially obfuscates development and might increase cost significantly. To overcome this problem virtual control devices can be introduced. Our paper has two main contributions. First, the design and implementation of a virtual game controller as an interface between any application in the context of video games and any kind of hardware input devices on multiple platforms are discussed. Second, a modification of the $1 gesture recognizer introduced by Wobbrock et. al. is presented that speeds up the recognition of two-dimensional gestures significantly. Finally, we demonstrate the usage of the gesture recognizer as a high-level input device for our multi-platform virtual controller.

1 Introduction

In the last forty years computer games have become more and more complex and their development cost have increased dramatically. Today, the development budget for a high-end multi-platform game can exceed more than 40 million dollar[1]. Besides the enormous advances regarding storytelling, graphics, physical simulation, and artificial intelligence various kinds of new input devices have been developed.

Especially Nintendo's *Wii Remote*, Sony's *Move*, and Microsoft's *Kinect* start to substitute the traditional keyboard, mouse, and gamepad interaction metaphors. Considering the pecularities of all those different input devices during the development of a cross-platform game can become a very complex and cost-intensive task. To handle this complexity some toolkits have been developed. With the *Input Configurator Toolkit* [4] and the *VoodooIO gaming kit* [23] special toolkits for game development were presented to support controlling an application which uses different input devices. But none of these toolkits is flexible enough to handle all different kinds of versatile game interactions. Additionally, many games use complex input metaphors based on gestures, *combos* (special sequence of pushed buttons), or audio. Especially controlling games using gestures become more and more widespread. This enforces the need for a flexible virtual game controller that can work with traditional input devices like e.g. mouse or keyboard as well as complex control metaphors based e.g. on gestural or audio input.

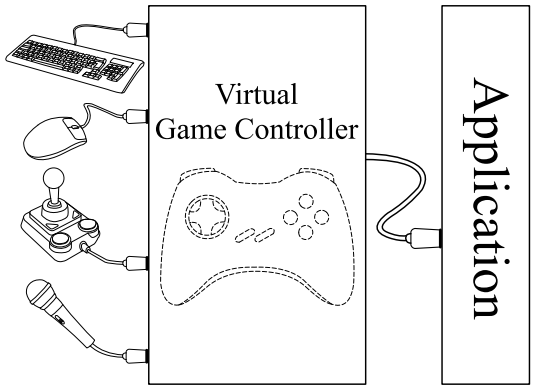

Fig. 1 Concept of a virtual game controller as an interface between any kind of hardware devices and any kind of applications.

In this paper we present the concept of a virtual game controller abstracting from all kind of real hardware controllers and input metaphors (see figure 1) admitting the development of multi-platform video games without the necessity to take care of all the different hardware devices. Today a single function in a computer game often can be controlled by many different input devices. For a detailed survey of the development of game input methods see [22]. Many of the new input devices empower the user to interact with an application by gestures in any kind. Our concept is flexible enough to handle varying real hardware devices as well

Andreas Schmidt, Ulrich Schwanecke
Dept. of Design, Computer Science and Media
RheinMain University of Applied Sciences
E-mail: kellendil@gmx.de, ulrich.schwanecke@hs-rm.de

Friedrich Volmering, Marc Kamradt
E-mail: {friedrich.volmering, mkamradt}@gmx.de

[1] http://www.develop-online.net/news/33625/Study-Average-dev-cost-as-high-as-28m, Retrieved August 14, 2011

as *high-level devices* such as gestures. Our system builds a layer between any kind of hardware input device and the application that allows to develop a computer game independently from the actual controller hardware. In fact, our system even allows to use very different hardware devices or input metaphors, like e.g. a keyboard and gestural input to control one and the same functionality.

We demonstrate the capability of our system by integrating a gesture recognizer as an input device that alternatively can be controlled by a standard keypad. As gesture recognizer we present an improvement of the 2D single stroke gesture recognizer for pen-based input devices which were first presented in [25] and later generalized for recognizing 3D gestures in [9]. Of course, also other methods for recognizing gestures are conceivable e.g. methods based on the measurement of accelerations that become interesting with the proliferation of smartphone equipped with accelerometer [11,15,1]. But no matter how a dedicated gesture recognizer is implemented it always can be integrated in our virtual controller concept in the way demonstrated below.

2 Virtual Controller

We designed the virtual controller as a configurable interface between an application and all possible input devices (see figure 2). Thereby, our design is based on a detailed requirements review. In the following we give a description of the component requirements that determine the design of our controller system as well as detailed discussion of the controller's architecture.

Our concept is based on a categorization of all possible input devices. We assign every real input component to a virtual component of one of the three categories *button*, *axis*, or *abstract*. Thereby

Buttons includes all types of physical buttons e.g. on joysticks, d-pads or keyboards.

Axis describes any kind of analog input devices as e.g. steering wheels, pedals, mices, or acceleration sensors.

Abstract contains all complex input metaphors like e.g. gestures, combos, or audio input.

The following section discusses the main requirements of the three described component categories.

2.1 Component requirements

Buttons are the simplest input components. In computer games it is often essential to distinguish between a button which is *pressed* and a button that is *typed*. To take this into account, a virtual button has to store the state it had in the last frame. If one now sets

$$l = \text{"button pressed in last frame"}$$
$$a = \text{"button pressed in actual frame"}$$

all possible states of a button can be described by the following boolean operations:

$$\bar{l} \wedge a = \text{"button } \textit{typed}\text{"}$$
$$l \wedge a = \text{"button } \textit{pressed}\text{"}$$
$$l \wedge \bar{a} = \text{"button } \textit{released}\text{"}$$
$$\bar{l} \wedge \bar{a} = \text{"no button activity"}$$

Some hardware devices (e.g. the XBOX 360 controller) have pressure sensitive buttons. To realize this hardware a virtual button has to store a float value between 0 (*"not pressed"*) and 1 (*"fully pressed"*). Additionally, two thresholds t_1, t_2 can be stored to simulate a button hysteresis, i.e. the button does not enter the state *"pressed"* until its value is above t_1 and it does not switch from *"fully pressed"* to the state *"released"* until its value is below t_2.

Axis can be separated in *half axis* and *full axis*. Half axis are padels used in car simulations for example and are represented by values in $[0, 1]$. In contrast, full axis are steering wheels for example and are represented by values in $[-1, 1]$. In practice, both types of axis often have a configurable *deadzone* as well as a bending-streching-factor manipulating its deflection. The latter for example allows precise controlling with small deflections of a joystick and a fast acceleration of a gaming figure with a quick deflection of the joystick.

Abstract components are highly dependent on the actual hardware device. Therefore, we consider an abstract component as a self-contained unit that is presented to the application as a button or an axis. The application then just needs to know the state of this button or axis respectively.

2.2 Signal Processors

To realize one and the same functionality with different hardware devices the virtual devices of our virtual controller can be manipulated by *digital signal processors (DSP)* allowing to simulate any kind of device behavior. Through this concept for example it is possible to simulate a steering wheel with just two digital hardware buttons. If one button is pressed the DSP constantly decreases a value which simulates a steering movement to the left. If the other button is pressed the DSP increases the value which simulates a steering movement to the right. That means that an appropriate DSP allows to simulate an axis with two buttons.

Another example for a DSP could be a *button repeat processor* simulating the reiterative pressing of a control button. Our system allows to interconnect any number of DSPs in any order. Therefore a device input can be manipulated arbitrarily allowing e.g. interpolation, invertation, negation etc. of an input signal.

2.3 Architecture

The main component of our system is the *virtual controller* which provides the application with virtual components (see figure 2). The virtual controller can be configured flexibly. The configuration describes the type of input signals expected by an application and the controller's behavior.

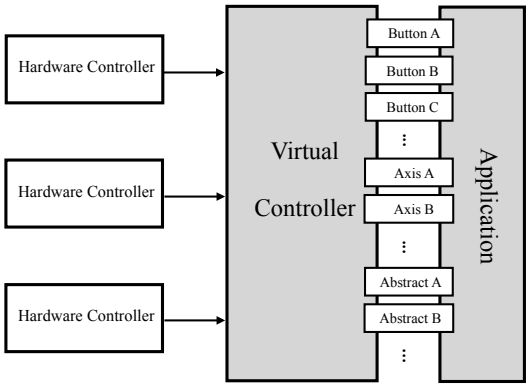

Fig. 2 Architecture of the virtual controller interface.

Beside the virtual controller there are *hardware controllers* which can be registered by the virtual controller. Thereby each hardware controller stores a device-ID identifying the hardware device and an object-ID identifying the input element (button, axis, ...) of the device. A pair of device-ID and object-ID is called *slot* and each virtual component can obtain several slots. Therefore our system allows a many to many mapping between virtual and real input devices. This makes it suitable to map several different inputs to the same functionality – something that is very important in modern computer games. Our system also provides persistent individual controller settings and comprehensive error handling.

2.4 Capturing

The virtual controller contains a capturing functionality facilitating its usage. To connect a virtual button with an input the virtual controller can be set into a capturing-state. In this state the virtual controller looks for the input of any connected hardware device and assigns the first input to the virtual button that should be configured. Thereby the virtual controller for example checks if there are any conflicts with other virtual buttons connected to the same device.

2.5 Error handling

An essential component of our virtual controller is error handling. An application must be able to react in a controlled manner if the connection to a hardware device gets lost e.g. due to a low battery of a wireless component or an accidentally disconnected plug between the input device and the computer or game pad. Therefore the virtual controller iterates over all slots in a configurable manner querying the hardware controllers if the slots are available. Now an application can query the virtual controller and react in a desired way if a needed control device is not available or the connection has been lost.

3 Gesture Recognition

Conceptually gesture recognition applications can be divided into two categories differentiating *pre-defined* and *user-defined* gestures [8]. In pre-defined gesture applications the system provides a fixed list of gestures whereas user-defined gesture applications empowers the users to define their own gestures. In both application types finally a list of gesture templates exists and the recognition of gestures is done by comparing the user input with this list of gesture templates.

In the scope of this report we are interested in recognizing 2D gestures on pen-based input devices. It is impossible to do justice to the numerous publications in gesture recognition as it would require citing several dozens of relevant papers. From the beginning of pen-based devices like e.g. PDAs, tablet PCs, or Whiteboards there is a hugh interest in recognizing handwritten user input [2,7,10,12,16,24]. A generic and extensible formal language to describe gestures is presented in [5]. Instead of trying to give a comprehensive overview we just glimps the different approaches of capital importance. The main ideas to solve the problem of recognizing gestures are based on dynamic programming [21], neuronal networks [18], hidden markov models [20], feature-based statistical classifiers [3], and pattern matching [21,13,25].

The gesture recognizer we present in this paper enhances the pattern matching approach called *$1 recognizer* which was introduced in [25]. The $1 recognizer is a very stable and efficient method to recognize 2D

gestures created with pen-based input devices. Originally it is a single stroke algorithm that is discussed in many subsequent papers like e.g. [14]. An extension to multistroke gestures can be found in [6]. To indicate our algorithm as a successor of the original one we entitle it *$1.5 gesture recognizer*.

Like the original $1 gesture recognizer our recognizer essentially consists of the following four steps:

1. **Resampling:** The tracked path of an user input is resampled to a fixed number of input points. In practice 32, 64, or 128 sampling points are enough to obtain satisfying recognizing results, depending on the complexity of the gestures to distinguish and the hardware capabilities of the target platform.
2. **Rotation:** To obtain a rotational invariant description of a gesture the input data are rotated by an 'indicative angle'.
3. **Scaling and translation:** To obtain a robust translational invariant gesture description the barycenter of the input gesture path is translated to the origin and afterwards non-uniformly scaled so it fits into $[-1, 1]^2$.
4. **Recognition:** Distances between the transformed path of the input gesture and all template gestures are calculated. The template gesture with the lowest distance to the input gesture is the recognized gesture.

The first improvement of the original $1 recognizer concerns step one of the algorithm. The original resampling step sometimes produces unexpected results, especially if a user draws the input gesture very fast. See figure 3 for an example where the original resampling algorithm produces a gesture path that is too short and therefore does not reproduce the input data in a proper way. The result of our new resampling algorithm (algorithm 1) is depicted on the bottom of figure 3.

Our second improvement concerns step four of the recognition algorithm. We introduce a distance map allowing us to decide on the basis of only a few point comparisons that an input gesture cannot match a given template gesture. Additionally we modified the original comparison algorithm between the user input and the templates resulting in a speedup up to 30%. This allows our recognizer to run on plattforms with very limited hardware capabilities like e.g. the Nintendo DS or low-end smartphones.

3.1 Modified resampling

The original $1 resampling algorithm iterates over the number of input points. This approach generates wrong resampling results if the number of input points is smaller than the desired sampling rate. Figure 3 depicts this phenomenon. The first row shows a user input consisting of six points that should be resampled with a sampling rate of 16 points. The original $1 resampling algorithm iterates over the six input points, copies the first four points, and generates two new points resulting in a set of points uniformly distributed on a part of the original input polygon. The result of this process is shown in the middle of figure 3. The resampled path obviously does not represent the user input in a proper way.

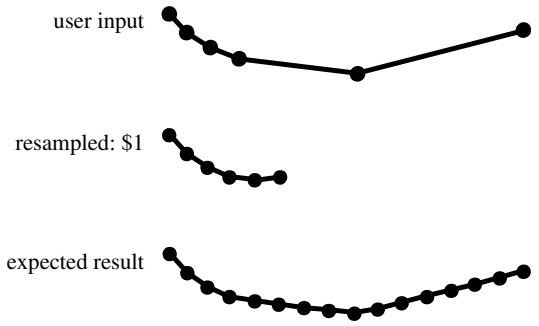

Fig. 3 The resampling algorithm of the original $1 recognizer results in wrong sampling if the user input contains less data than the sampling rate.

The expected result consisting of 16 points uniformly distributed on the input path is depicted on the bottom of figure 3. To obtain this result, we first check in our modified resampling step if the number of input points is smaller than the desired sampling rate. If the number of input points exceeds the sampling rate we use the original $1 resampling algorithm. If the number of input points is smaller than the sampling rate we use algorithm 1 to resample the input gesture. The algorithm iterates over all line-segments of the input polygon and resamples them if they are long enough. If line-segments are too short they are skipped and their lengths are added until the new segment is long enough.

3.2 Early-Out-Algorithm

In the recognition step of the original $1 algorithm for each gesture template a number of path-distances are calculated and a minimal path-distance using a *golden section search* [19,25] is determined. A path-distance is calculated by comparing all points of the user input and a template. Thereby always the complete path of a gesture is reconsidered which is not necessary in all cases. On the other hand, sometimes gestures start with a similar path and differ only much later. As an example consider figure 4. The two gestures $g1$ and $g2$

Algorithm 1: Resampling in case that the amount of user input data is below the actual sampling rate.

```
ResampleUpwards(points, sampleRate)
    I ← Pathlength(points) / (sampleRate − 1);
    rest ← 0;
    oldI ← 1;
    nPoints ← Length(points);
    Insert(newPoints, 0, points[0]);
    while Length(newPoints) < sampleRate do
        d = Distance(points[oldI − 1], points[oldI]);
        if (d + rest) ≥ I then
            if rest == 0 then
                m ← I;
            else
                m ← abs(I − rest);
            r ← (points[oldI] − points[oldI − 1])/d · m;
            q ← points[oldI − 1] + r;
            Append(newPoints, q);
            Insert(points, oldI − 1, q);
            rest ← 0;
        else
            if oldI < nPoints − 1 then
                rest ← rest + d;
                oldI ← oldI + 1;
            else
                /* catch rounding errors */
                Append(newPoints, points[nPoints − 1]);
    return newPoints;
```

are identical in their first third. Comparing the first i points of both gestures would state that they are identical. But comparing two points of both gestures with index $i + j$ immediately shows that both gestures are different.

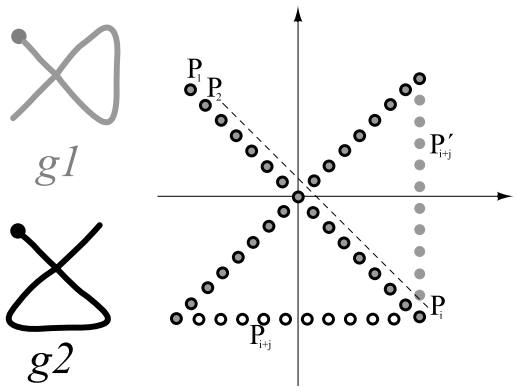

Fig. 4 Gestures $g1$ and $g2$ are congruent on the first third of their path. A comparison of the two path discovers a difference between the two path not until the first third of the path is completely compared.

In order to efficiently check which points should be tested we first generate an index-list in a preprocessing step. The index-list describes which points should be compared first and is constructed as follows:

To every point P_k^i of a gesture template a global mean distance ϕ_k to every point P_k^j of the other gesture templates is calculated as

$$\phi_k = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left\| P_k^i - P_k^j \right\|.$$

The indices of the points are sorted downwards based on the value of ϕ_k. Comparing input points with template gestures ordered based on ϕ_k ensures that points which are furthest apart are compared first. This approach enables an early decision if an user input and a gesture template are different. Therefore we named this approach *Early-Out-Algorithm*.

Additionally, we introduced two parameters. The first parameter EOT (Early-Out-Threshold) is a threshold for the distance between an input point and a template point. If the distance is greater than EOT an input point and a template point cannot match. The second parameter EOC (Early-Out-Chances) determines the acceptable number of distances that are larger than EOT.

3.3 Integration into the virtual controller

To integrate the gesture recognizer into the virtual controller we implemented a virtual component for abstract inputs. The input of this component is a point stream representing the user's gestural input. The component realizes a gesture recognition based on a set of preloaded gesture templates. Each of this templates represents a virtual button that is triggered if the relevant gesture is recognized (see figure 5).

As mentioned above a button can have several slots. Therefore beside gestures other devices like e.g. real hardware buttons on a keyboard or gamepad can also trigger the buttons activated by the gesture recognizer.

4 Evaluation

We evaluated our improved gesture recognizing algorithm based on two sets of test gestures. Template-Set 1 is the set of templates described in [25] presenting the $1 gesture recognizer. The depicted gestures are very simple and can easily be distinguished. The gestures in Template-Set 2 are more complex. They are used in the Nintendo DS game *Lost Magic*[2].

[2] Taito-Ubisoft, 2006

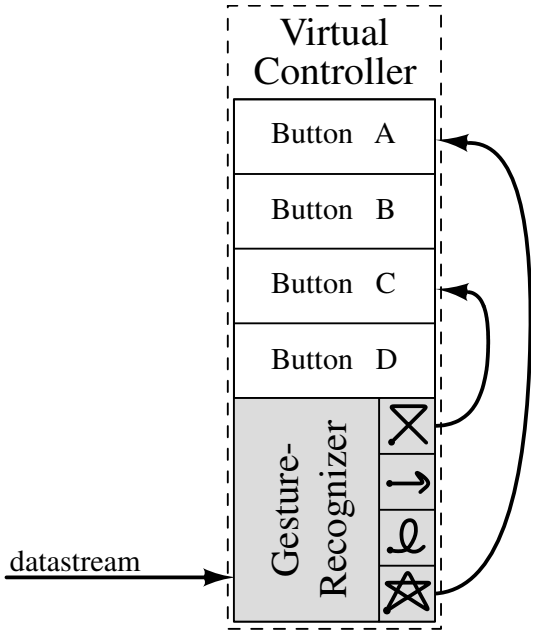

Fig. 5 Integrating the recognizer into the virtual controller.

We measured the time that is needed to recognize a gesture after the user input. Thereby we tested different values for EOC using a fixed value of 0.2 for EOT. The results are shown in table 1. Table 2 shows the same test with the error tolerance doubled i.e. EOT=0.4.

The data shows that tolerating more errors (increasing EOC) results in an increased recognition time. For an EOC larger than the sampling rate and a large EOT the Early-Out-Algorithm uses all points just like the original $1 algorithm and therefore has the same running time. The smaller EOC becomes the faster the new algorithm is but at the same time the recognition rate decreases. If the value of EOT increases the algorithm on the one hand becomes more tolerant against scribbled gestures but on the other hand becomes slower.

Suitable values for EOC and EOT depend on the shape of the gestures (whether the gestures are easy to draw or not) as well as on the user interface that is applied to produce the gestures. An interface for example that guides the user carefully and gives him hints on how to draw a special gesture would result in accurate input gestures that can be recognized reliably even with EOC set to small values. In contrast, if the user has to draw a gesture on a plain surface the input would be more jittered and the value of EOC has to be chosen higher in order to obtain a reliable gesture recognition.

To summarize EOC and EOT can be used to configure the gesture recognizing process to anything between a fast but less accurate and a slower but more accurate system. The explicit choice of values for EOC and EOT heavily depends on the concrete application.

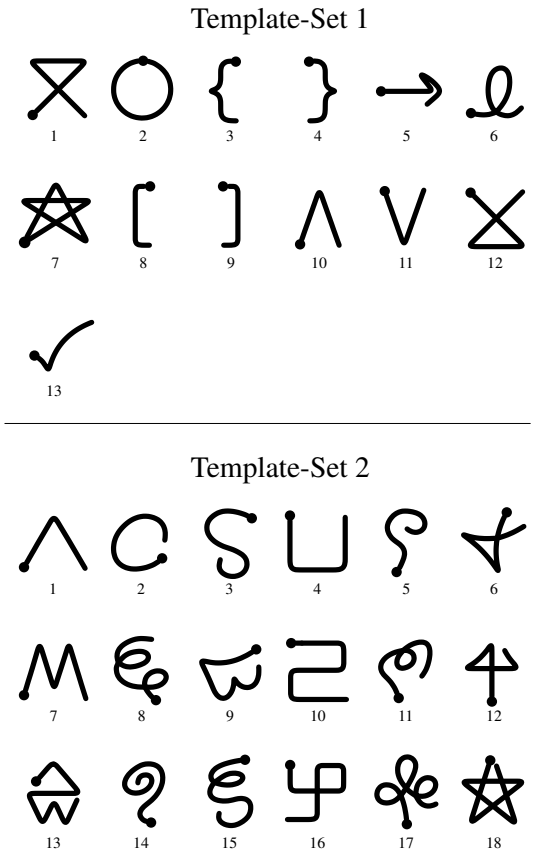

Fig. 6 Test set of templates. Template-Set 1: Gestures used to evaluate the original $1 recognizer. Template-Set 2: Gestures of a Nintendo DS game (Lost Magic, Taito-Ubisoft, 2006).

Table 3 and 4 give a comprehensive comparision of the original $1 recognizer and our improved gesture recognizing algorithm. We tested the two different sets of gesture templates depicted in figure 6 with different sampling rates. In table 3 the results for the original Template-Set 1 are listed. It can be seen that our Early-Out-Algorithm always is faster than the original $1 recognizer. Depending on the sampling rate our algorithm results in a speedup of more than 30%. For the more complex Template-Set 2 the Early-Out-Algorithm also is always faster than the original $1 recognizer. Depending on the sampling rate the speedup varies between 20-30%.

5 Conclusion and Future Work

In this paper we presented a flexible virtual controller that can virtualize any kind of input device and therefore make game development more independent from the emerging various new kinds of input devices. Our system can abstract simple devices like e.g. joysticks as well as more complex ones like e.g. gestures.

EOT: 0.2 Gesture	%	EOC: 0 ms	1 ms	3 ms	5 ms	7 ms	9 ms
Template-Set 2 - Sample Rate: 32							
2	73	-	-	33	35	36	37
7	80	-	-	33	34	35	37
9	67	-	-	-	35	36	37
13	66	-	-	-	34	35	36
17	81	-	-	33	34	35	36
Template-Set 2 - Sample Rate: 64							
1	76	-	-	-	-	64	65
4	85	-	-	62	63	64	66
12	76	-	-	-	63	65	66
14	86	60	60	61	62	64	65
18	78	-	-	-	-	-	65

Table 1 *Recognition time with EOT=0.2 and different values of EOC (number of tolleranted erros). Column* Gesture *depicts the gesture of the respective Template-Set shown in figure 6. Column* % *depicts the accuracy with which the gesture is drawn. Compare also with table 2.*

EOT: 0.4 Gesture	%	EOC: 0 ms	1 ms	3 ms	5 ms	7 ms	9 ms
Template-Set 2 - Sample Rate: 32							
2	59	-	-	35	37	39	41
3	69	-	-	33	35	36	38
10	63	32	32	33	36	38	39
12	51	-	-	36	38	40	41
17	56	32	32	33	35	37	39
Template-Set 2 - Sample Rate: 64							
5	56	62	62	65	67	69	71
7	51	-	61	62	63	65	66
8	64	64	64	65	66	67	69
15	52	60	60	62	63	65	66
18	52	62	62	65	67	69	71

Table 2 *Recognition time with EOT=0.4 and different values of EOC (number of tolleranted erros). Column* Gesture *depicts the gesture of the respective Template-Set shown in figure 6. Column* % *depicts the accuracy with which the gesture is drawn. Compare also with table 1.*

We demonstrate the capability of our system by implementing an improved gesture recognizing algorithm. Our algorithm is based on the $1 recognizer presented by Wobbrock et. al. but improved in two ways. First we developed a new resampling algorithm which overcomes the problem of the original algorithm when the number

Gesture	%	$1-Recognition Time (ms)	EarlyOut-Recognition EOT/C: 0.3/4 Time (ms)	Diff (ms)
Template-Set 1 - Sample Rate: 32				
1	83	35	27	8
2	93	36	28	8
4	78	35	27	8
7	75	35	27	8
13	95	35	27	8
Template-Set 1 - Sample Rate: 64				
3	82	67	50	17
5	74	66	47	19
6	72	67	49	18
9	89	68	50	18
10	97	68	49	19
Template-Set 1 - Sample Rate: 128				
2	96	131	88	43
7	64	131	89	42
8	94	131	93	38
11	91	131	89	42
12	93	131	89	42

Table 3 *Comparison of the original $1 recognizer and the Early-Out-Algorithm using Template-Set 1and different sampling rates. Column* Diff (ms) *shows that the Early-Out-Algorithm allways is faster than the original $1-recognizer.*

of input points is less than the sampling rate. Second, we presented an Early-Out-Algorithm that speeds up the recognition time up to 30% compared to the original recognition algorithm.

Our concept of a virtual controller is very flexible and allows an easy integration of any kind of emerging new input devices even very special ones e.g. like the device described in [17]. It can provide a basis for commercial game development as well as the development and evaluation of new input devices e.g. like the one described in [22].

Our actual implementation is frame based and input signals are not cached in any kind which could result in data loss. To fix this and to integrate support of force feedback devices will be the natural next steps. Thereby integrating force feedback devices requires to additionally submit information from the application to the input devices. Furthermore, we are planing to integrate logging functionality into our system to allow to record and replay user inputs.

Gesture	%	$1-Recognition Time (ms)	EarlyOut-Recognition EOT/C: 0.3/4 Time (ms)	Diff (ms)
Template-Set 2 - Sample Rate: 32				
1	72	49	37	12
2	90	48	38	10
3	93	48	37	11
4	83	49	37	12
5	92	48	37	11
11	85	48	37	11
12	61	48	38	10
13	82	48	36	12
14	87	48	36	12
Template-Set 2 - Sample Rate: 64				
6	69	93	67	26
7	78	92	65	27
8	94	93	66	27
9	75	93	65	28
10	84	93	65	28
15	76	93	65	28
16	76	93	65	28
17	87	93	65	28
18	84	92	67	25

Table 4 *Comparison of the original $1 recognizer and the Early-Out-Algorithm using Template-Set 2 and different sampling rates. Column* Diff (ms) *shows that the Early-Out-Algorithm always is faster than the original $1-recognizer.*

References

1. Sandip Agrawal, Ionut Constandache, Shravan Gaonkar, Romit Roy Choudhury, Kevin Caves, and Frank DeRuyter. Using mobile phones to write in air. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 15–28, New York, NY, USA, 2011. ACM.
2. X. Cao and R. Balakrishnan. Visionwand: Interaction techniques for large displays using a passive wand tracked in 3d. pages 173–182. Proc. UIST '03. New York: ACM Press, 2003.
3. M. G. Cho. A new gesture recognition algorithm and segmentation method of korean scripts for gesture-allowed ink editor. pages 1290–1303. Information Science 176 (9), 2006.
4. P. Dragicevic and J. Fekete. The input configurator toolkit: towards high input adaptability in interactive applications. In *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, pages 244–247, 2004.
5. F. Echtler, G. Klinker, and A. Butz. Towards a unified gesture description language. In *Proceedings of the 13th International Conference on Humans and Computers*, pages 177–182, Fukushima, Japan, 2010. University of Aizu Press.
6. M. Field, S. Gordon, E. Peterson, R. Robinson, T. Stahovich, and C. Alvarado. The effect of task on classification accuracy: using gesture recognition techniques in free-sketch recognition. In *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '09, pages 109–116, New York, NY, USA, 2009. ACM.
7. F. Guimbretire, M. Stone, and T. Winograd. Fluid interaction with high-resolution wall-size displays. pages 21–30. Proc. UIST '01. New York: ACM Press, 2001.
8. F. Guo and S. Chen. Gesture recognition techniques in handwriting recognition application. In *Proceedings of the 2010 12th International Conference on Frontiers in Handwriting Recognition*, ICFHR '10, pages 142–147, Washington, DC, USA, 2010. IEEE Computer Society.
9. N. Haubner, U. Schwanecke, and R. Dörner. Recognition of dynamic hand gestures with time-of-flight cameras. In *Sensysble Workshop 2010, ITG/GI Workshop on Self-Integrating Systems for Better Living Environments*, pages 33–39. Shaker Verlag, 2011.
10. K. Hinckley, G. Ramos, F. Guimbretiere, P. Baudisch, and M. Smith. Stitching: Pen gestures that span multiple displays. pages 23–31. Proc. AVI '04., ACM Press, 2004.
11. M. Jang, J. Kim, J. Sohn, and H.S. Yang. Design of an adaptive accelerometer-based handwriting recognition system based on metacognitive framework. In *Consumer Electronics (ICCE), 2011 IEEE International Conference on*, pages 175–176. IEEE, 2011.
12. A.K. Karlson, B.B. Bederson, and J. SanGiovanni. Applens and launchtile: Two designs for one-handed thumb use on small devices. pages 201–210. Proc. CHI '05. New York: ACM Press, 2005.
13. P. Kristensson and S. Zhai. Shark2: a large vocabulary shorthand writing system for pen-based computers. In *UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 43–52, 2004.
14. P. O. Kristensson and L. C. Denby. Continuous recognition and visualization of pen strokes and touch-screen gestures. In *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '11, pages 95–102, New York, NY, USA, 2011. ACM.
15. J. Liu, L. Zhong, J. Wickramasuriya, and V. Vasudevan. uwave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive Mob. Comput.*, 5:657–675, December 2009.
16. M.R. Morris, A. Huang, A. Paepcke, and T. Winograd. Cooperative gestures: Multi-user gestural interactions for co-located groupware. pages 1201–1210. Proc. CHI '06. New York: ACM Press, 2006.
17. Y. Nakatsuru, R. Hiramatsu, H. Mori, and J. Hoshino. A soft body controller that can be thrown and grasped. In *VR Innovation (ISVRI), 2011 IEEE International Symposium on*, pages 333–334. IEEE, 2011.
18. J. A. Pittman. Recognizing handwritten text. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 271–275, 1991.
19. W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. Numerical recipes in c. Cambridge Univ. Press, 1992.
20. T. M. Sezgin and R. Davis. Hmm-based efficient sketch recognition. In *IUI '05: Proceedings of the 10th international conference on Intelligent user interfaces*, pages 281–283, 2005.
21. C.C. Tappert. Cursive script recognition by elastic matching. pages 765–771. IBM J. of R. & D. 26 (6), 1982.
22. A. Thorpe, M. Ma, and A. Oikonomou. History and alternative game input methods. In *Computer Games (CGAMES), 16th International Conference on*, pages 76–93. IEEE, 2011.
23. N. Villar, K. M. Gilleade, D. Ramdunyellis, and H. Gellersen. The voodooio gaming kit: a real-time adaptable gaming controller. *Comput. Entertain.*, 5(3):7, 2007.
24. A.D. Wilson and S. Shafer. Xwand: Ui for intelligent spaces. pages 545–552. Proc. CHI '03. New York: ACM Press, 2003.
25. J. O. Wobbrock, A. D. Wilson, and Y. Li. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *UIST '07*, pages 159–168, 2007.