Gestures, Shapes and Multitouch Interaction

Augusto Celentano and Andrea Minuto
Dipartimento di Informatica, Università Ca' Foscari Venezia
via Torino 155, 30172 Mestre (VE), Italy
{auce,aminuto}@dsi.unive.it

***Abstract*—We discuss issues related to the design of a multitouch gesture sensing environment, allowing the user to execute both independent and coordinated gestures. We discuss different approaches, comparing frontal vs. back projection devices and gesture tracking vs. shape recognition. To compare the approaches we introduce a simple gesture language for drawing diagrams. A test multitouch device built around FTIR technology is illustrated; a vision system, driven by a visual dataflow programming environment, interprets the user gestures and classifies them into a set of predefined patterns, corresponding to language commands.**

I. Introduction

Natural interaction [1]–[3] has received great attention in last years, due to progress in research about interaction devices and to new applications, mainly in the Web2.0 area. Ubiquitous computing and ambient intelligence compel us to rethink our relations with information devices, trying to surpass the traditional "window, icon, menu, pointing device" (WIMP) style of interaction with multimodal and distributed interfaces.

The WIMP paradigm is, however, so well rooted that many efforts to find more natural interaction styles through free gesture interpretation are still based on the model of a pointer moving in a limited area, commanding actions by "clicking" on objects representing programs and documents. Indeed, the 2D desktop layout doesn't leave too much space to freely propose "real world" gestures, possible only in an immersive environment and with constraints.

In this paper we argument about the choice of a gesture sensing device and the design of a gesture language for it. We discuss the pros and cons of different interaction styles, and their impact on the gesture structure and interpretation. We define a drawing-style gesture language suited for a multitouch device based on the Frustrated Total Internal Reflection (FTIR) technololgy proposed by Han [4], allowing users to interact with real multitouch, independent gestures. The gesture language is targeted to drawing simple graphs such as the ones used in conceptual and mental maps. The operations permitted are: create an object (a node), move an object, connect two or more objects, write text labels, delete objects.

We have built a prototype system to experiment our proposal. Gesture are captured and interpreted by a vision system built around a dataflow programming environment; gestures are classified into a set of predefined patterns corresponding to the language commands. Having multiple pointers allows the user to execute both parallel independent gestures (e.g., to select many objects at the same time), or coordinated gestures (e.g., to connect two objects).

The paper is organized as follows. After reviewing a few recent proposals in multitouch devices in Section II, we discuss the relationships between gestures, user perception and sensing environment in Section III. Section IV compares gesture vs shape recognition, justifying our choice to rely on the latter. Section V discusses the gesture language design. The prototype is presented in Section VI. Section VII draws the concluding remarks.

II. Gestures and multitouch devices

In the context of this paper a multitouch device is a surface where multiple active contact points can be sensed at the same time, which is able to acquire touch information at reasonable spatial and temporal resolution. Our interest is oriented to devices scalable up to large installations, apt to support collaborative as well as personal applications.

Among the different interface types suited for gesture based interaction two main classes can be identified: interfaces based on pressure sensitive hardware and interfaces based on vision systems, recognizing the user gestures in a 3D space or in a 2D projection of it. A third class of systems uses a pressure sensitive surface, but the gestures are tracked by a vision system revealing and interpreting the user touch on the surface. While in principle gesture languages are independent of the interface technology, a number of constraints are imposed by the choice of a specific class.

Actually, interfaces based on pressure sensitive hardware such as the DiamondTouch [5], the Apple iPhone/iPodTouch, and Lemur, a control surface for audio device, based on a patented multitouch technology [6], restrict the number of simultaneous touch points sensed. More general solutions can be programmed based on vision systems able to track the position of a variable number of contact points. In most cases touch is revealed by a FTIR device [4], where the number of simultaneous light spots is limited only by the device size, the spot size and the performance of the vision system. Microsoft Surface$^{\text{TM}}$, announced in 2007, is also based on a combination of FTIR and multi-camera sensing [7].

Some systems rely on recognizable markers to spot the user actions. MagicBoard [8] aims at improving the capabilities of a normal whiteboard by tracking the movement of the user pen, revealed by a marker and a camera. The movement is translated into an electronic copy of the user drawing. Markers of different colors allow several users to be tracked simultaneously.

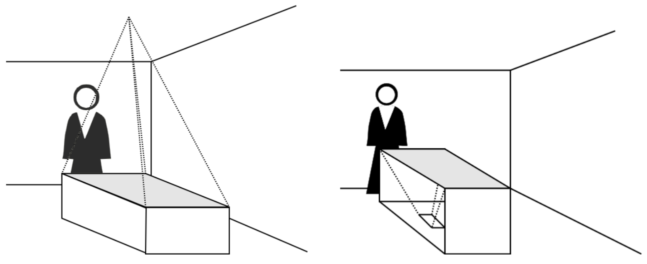

Fig. 1. Two vision based installations for gesture interaction: (a) frontal projection, (b) back projection

Reactable [9] is a back-projection system for the collaborative creation of electronic music performances on a multitouch tangible interface. Several performes can operate by manipulating marked physical objects representing the components of a synthesizer, which are tracked by a fiducial tracker following the markers exposed to an infrared light.

More complex vision systems are based on immersive environments, like virtual and augmented reality. HandVu [10] is an augmented reality system, composed of a camera placed on the user head to cover the visual filed of the user, able to track the user hands and some postures, to which specific meanings are assigned.

Dream space by IBM [11] is one of the first experiments to build a natural interface. Based on the interpretation of the user posture and hand gesture, it was aiming at putting the user at the center of the interaction space, but the project has been discontinued.

III. Sensing user gestures

According to Ishii and Miyake [12] one of the drawbacks of WIMP interfaces is the lack of visual correspondence between the user sight and the pointer. The user sees the effect of the pointer movement in a visual space different from the gesture space. Touch screens are a partial remedy to this situation, but the current technology limits their generality by alternatively constraining the precision (if using fingers, which allow multitouch gestures) or the gesture variants (if using a stylus, which can differentiate gestures only through timing and menu options). Designers have invented artifices to differentiate gestures and commands in these cases.

With reference to vision based systems, which do not limit the number of simultaneous pointers, the system perception of the user gestures is thus depending on the device installation. Two main cases occur: (a) a frontal projection approach, where the workspace image projector and the sensing camera are positioned in front of the interaction surface, tracking the position of the user's hands or fingers acting as pointers; (b) a back projection approach, where the workspace image projector and the sensing camera are behind the interaction table, usually built around the FTIR technology to reveal the touch point. Figure 1 illustrates two stereotypes of frontal and back projection.

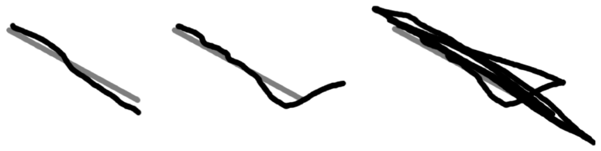

Fig. 2. Correcting a gesture

In the frontal projection configuration of Figure 1(a), the identification of the user pointer may be ambiguous, since part of the user body is in the working area. The user hands or fingers to be tracked need to be artificially marked, diminishing the natural interaction effect.

In the back projection configuration of Figure 1(b) there is no "hover" state, since the gesture is sensed only when the user is touching the sensible surface. Simmetrically, in the frontal configuration of Figure 1(a) there is no "touch" state (unless explicit markers are used), and the start of a gesture is implicit in *any* user movement. A feasible solution to overcome this problem is to rely on timing, e.g., assuming that "hovering" on an object for some time means to touch it, and staying still for a defined amount of time means the gesture end. Such artificial patterns, besides lessening the natural approach to interaction, are prone to errors and misunderstanding and cannot be used for robust gesture recognition.

We claim that back projection is to be preferred, lacking the feedback of the hover state but gaining in deterministic identification of a gesture start and end.

IV. Gesture vs. shape recognition

Independently from the specific application, a vision based interface must execute three basic functions: sensing, identifying, and tracking. *Sensing* is the action of perceiving the presence of an event or of a phenomenon: for example, there is a (moving) touch point. *Identifying* is the action of interpreting the event or the phenomenon; for example, the user is executing a gesture with such (geometric) features. *Tracking* is the result of observing the evolution of the phenomenon, computing its actual state and relating it with the past states; for example, the user has drawn a shape recognized by the system [13].

Recognizing a user gesture can be done by continuously interpreting the dynamics of the gesture, or by interpreting the shape traced, much like a drawing. Both have pros and cons.

A. Gesture interpretation

If the gesture meaning is defined by its dynamic evolution, variants in the execution might be interpreted as variants in the meaning. For example, drawing a shape clockwise or counterclockwise could lead to different results. The greater flexibility of the gesture language is paid in terms of possible misinterpretations and mistakes.

The intepretation of the gesture evolution implies that the interpretation takes place while the gesture is executed. Hence, the execution must be continuous and coherent. This process,

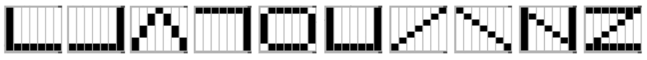

Fig. 3. The standard shapes recognized by the prototype

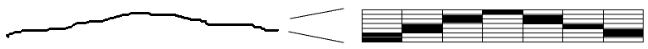

Fig. 4. An incorrect interpretation of an almost straight line

while deterministic at last, might go through phases where the user action could correspond to many different gestures. A simple example is the tracing of a curved line (an arc) that could also be interpreted as the initial tracing of a closed figure (e.g., a contour enclosing objects). The interpreter needs to face an intrinsic ambiguity in the gesture analysis, processing the gesture in two times: waiting until the gesture is finished to assign it to a specific class (e.g., an arc or a closed contour), then interpreting its dynamics such as the direction, the timing, the pauses, and so on, according to the class identified.

The dynamic interpretation of a gesture impacts also on the possibility to correct it. Since the complete evolution of the gesture is interpreted and not only the final trace, it is impossible to undo or to correct the gesture.

B. Shape interpretation

Shape recognition is easier both from the user and from the system perspective: the problems noted above can be overcome because recognition takes place after the gesture has been completed. On the negative side, lines and figure borders are not oriented, unless some visible convention is used to mark the initial point of contact; timing is also not relevant.

As the user action is interpreted only when it is complete, errors and uncertainty in the drawing can be corrected to some extent, e.g., by repeating part of a gesture. In Figure 2 the first shape represents a correct gesture, interpreted as a straight line; the second shape deviates from a straight line and could be interpreted erroneously. The user can correct it by insisting on drawing a more linear path, forcing the system to interpret it correctly.

We claim that shape recognition is to be preferred because it is more robust and easier to implement. In the context of the chosen domain a dynamic gesture analysis gives no real advantage. We feel also that the user, confirming the gesture only after completion, can act in a more comfortable way, even if a proof of our feeling will come only after a usability evaluation.

C. Shape analysis and recognition

The system starts the gesture analysis when a touch point is sensed. A dynamic bounding box is incrementally built around the gesture until the user stops touching. The bounding box is sampled into a 7x7 grid, whose cells are marked if crossed by the gesture trace.

Shape recognition is done against a set of predefined standard shapes, computing a similarity measure and selecting the most similar one, provided it is over a suitable threshold. Figure 3 shows the set of shapes recognized by the prototype implementation, as visually defined in the VVVV programming environment (see Section VI).

Two cases in the shape interpretation are handled as special cases: small figures and straight horizontal and vertical lines.

Due to the use of the fingers as pointing devices and to the physical properties of FTIR sensing, the light spots sensed have a dimension which prevents a correct interpretation of very small shapes. The sampled matrix would have most cells marked due to the relative size of the light spot and the trace. Therefore, gestures whose size is below a threshold are sensed as simple touch points and not as shapes.

Straight horizontal and vertical lines are handled as special cases, by demoting the shape to a line if the ratio between the longer and the shorter side of the bounding box is higher than a threshold. The reason for such a choice is related to the same constraints exposed above about the ability to discern touched from untouched cells in a small space. Figure 4 illustrates the problem: the (almost) horizontal line on the left could be interpreted as the "$\bigwedge$" gesture which activates a virtual keyboard (the third shape in Figure 3); since its elongation trespasses a threshold (actually a ratio of 3 between the sides) it is demoted to a horizontal line.

V. The interaction language

The interaction language is modeled on the statechart formalism, where each state represents a phase of gesture recognition, and the internals of the state describe how that phase is exploited. Three state classes are defined, borrowing the terminology from UML: pseudo-states marking the beginning and the end of the gesture recognition; description states, representing the persistence of a condition in the gesture execution (e.g., the user is tracing a line, or the user touch is over an object); superstates, grouping states with similar functions, thus representing interpretation processes at a coarser level of detail. States are connected by transitions which in principle can be time triggered or event triggered. We shall comment on time-triggered transitions later.

Due to the large number of gestures the user can do, among which only a few are meaningful for the application evolution, we make three assumptions: (1) each gesture is continuous; (2) the gesture start and end events can be unambiguously identified; (3) each gesture is traced while it is executed, but its shape is interpreted only after it has been completed.

These assumptions allow us to define only the statecharts for the gestures that are recognized as correct, and change the state of the system, while the gestures that are not explicitly recognized are simply discarded, returning the system to the state it had before the gesture start. Visually, that means that the trace projected by following the gesture on the surface while it is executed is deleted after completion; the trace of a recognized gesture is replaced by the new state of the interface, e.g., a node or an arc is drawn, or the virtual keyboard is

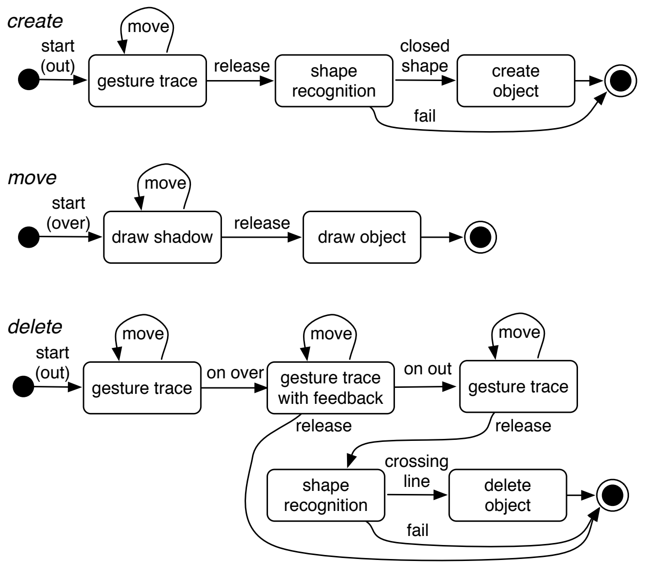

Fig. 5. The state diagrams for *create*, *move* and *delete* gestures

displayed. Indeed, the recognition process is very flexible and gives the user much freedom while getting a high rate of matches; discarding a shape is not so frequent in large gestures. Since small gestures are considered as points, the system must be calibrated according to the size of the panel.

The assumption (1) above is important because it simplifies the definition of the system states by identifying the starting pseudo-state with the user touch on the sensing surface. Gestures in a frontal projection system are more difficult to analyze, because the user hands are always in the viewfield of the capturing camera, and the start of a gesture cannot be associated to the act of touching. Hence the need, in general, to introduce also transitions based on time delays signaling the start and end of a user action.

Figure 5 illustrates three diagrams corresponding to the analysis of the *create*, *move* and *delete* gestures. The diagrams are simplified with respect to the prototype, showing only the most relevant events occurring in gesture analysis. The analysis is done in parallel on all the defined gestures, but at most one is recognized.

The top diagram illustrates the creation of a node, which occurs when the user draws a closed figure, e.g. a circle or a square. As long as the user touches the sensing surface the contact point is visually traced as a line. The *shape recognition* state, after the gesture has been completed (*release* event), interprets the shape and emits the *closed shape* event if the shape is similar to the fifth pattern of Figure 3, otherwise emits a *fail* event and goes to an idle state. The diagram is relevant if the user starts a gesture out of an object. The middle diagram describes the *move* gesture, started by touching an already created object; a shadowed object is drawn at the touch point as long as the user moves the finger. When the user releases touching, the object is drawn in the new place. The bottom diagram describes the *delete* gesture, which is made by "crossing out" an object with a straight line. The gesture

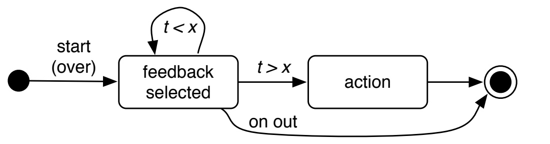

Fig. 6. The object selection diagram in frontal projection

is started, as in the creation case, by touching the sensing surface out of an object. If the trace crosses an object (*on over* event) the object is highlighted to give user a visual feedback, and tracing continues until the user releases the contact. If the contact is released when still on the object, the gesture is discarded, otherwise the shape is interpreted; if it is a straight line and crosses the object, then the object is deleted, otherwise (e.g., the line ends on the object) the gesture is discarded.

In a front projection installation environment, the assumptions made at the beginning of this section are no longer valid, since the user action is continuous and, in principle, there is no way to mark the start of a gesture as different from free movement and positioning. To give a flavour of the problems occurring in this scenario, in Figure 6 the state diagram of a generic object selection action is shown; the selection occurs if the user "points" to the object for some time t longer than a threshold x, $t > x$. The action is started when the system recognizes the user hand or finger over the object (*start (over)* event). While the user stays on the object nothing happens until exceeding the time threshold x, then the action is executed. If the user leaves the object before the threshold time has elapsed, the system returns to an idle state.

VI. A PROTOTYPE

We have built a prototype around a home-made FTIR sensing plate of plexiglas, not different from the one described by Han in [4], using a low cost webcam modified to capture only infra-red light. The goal of the prototype, besides exercising the basic gestures defined to evaluate issues such as ambiguity and size limits in shape recognition, was not to test the hardware, which doesn't present features worth to be discussed here. Indeed, the main goal was to test a visual dataflow programming (VDP) environment, namely the VVVV toolkit [14], designed for audio and video real-time processing, to analyze and interpret the gestures without developing own tracking algorithms. The VDP environment of VVVV (like Max/MSP [15], well known and widely used by multimedia performers) is based on the iterative and timed processing of *frames*, which are the states of multimedia information (video in our system, audio and video in general) processed by devices like filters, comparators, multiplexers, etc., called *patches*; patches correspond to the functions of a conventional programming language, visually connected by arcs representing signal flow paths.

VVVV is free for non commercial use; it has been chosen for its video processing capabilities and because a large community of users exists, providing patches for several applications. The main drawbacks of VVVV with respect

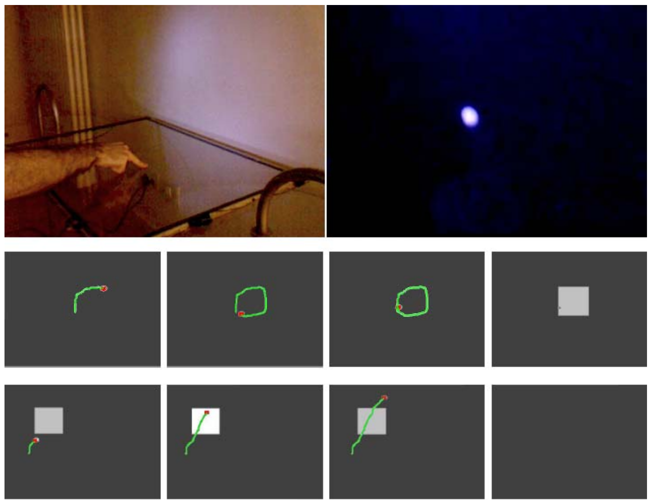

Fig. 7. The prototype multitouch panel (top), the *create* (middle) and *delete* (bottom) gesture interpretation

to other products, like Max/MSP, are a weaker performance and the availability only in the Windows platform. Both such drawbacks are not relevant for our goal.

The experiment has been successful in demonstrating the advantages of a VDP environment: it eases the development of vision based applications; it naturally handles the "multi" part of user touch and interaction; it fits well the state based design of the interaction language; finally, it is also well suited for rapid prototyping, due to its fully interpreted behaviour.

Due to the limited size of this paper, only a small example of the prototype action can be illustrated. In Figure 7 the two top images show the multitouch panel prototype and the infrared image sensed by the camera when the user touches the surface. The middle four images show snapshots of the gesture interpretation output by VVVV. As the user releases the contact with the panel, the light spot of the touch point disappears and the interpreter evaluates the shape. Since it is a closed figure similar to the "O" shape in Figure 3 it is interpreted as a node creation command, and the bounding box of the trace is drawn.

The bottom four images of Figure 7 illustrate the node deletion gesture, executed by crossing out an object with a line. When the contact is released the shape of the gesture is evaluated: if it is an open figure and crosses an object by traversing its opposite sides, then a delete gesture is recognized, and the object is deleted.

VII. Conclusion

We have discussed the design of a sensing environment and a gesture language aiming at introducing the bases for the development of natural interaction systems. The prototype needs to be tested to reveal its pros and cons, to drive further investigation on this field, but some outcomes are already perceivable.

The hardware quality plays a role in the ability to move from a demonstration of feasibility to a usable product. The large dimension of the light spot, a consequence of the FTIR technology evident in Figure 7, is a serious drawback for applications with high density of information on the interaction panel, which is only partly balanced by the low cost of the installation. Hence, precision in the touch position is hard to obtain. Such a limitation prevents the designer from developing real drawing applications, favoring instead the use of such interfaces for interacting with object representations; it is a step towards the tangible interfaces, where the objects are usable *per se* and not through their virtual representations. The lack of precision is known in all the interfaces based on the use of hands and fingers as pointers, and leads the designers to introduce artificial gestures such as pointing at the two sides of a central point which is the real target, as done, for example, in the DiamondTable applications.

The use of VVVV as a prototyping environment has been demonstrated highly effective due both to the interpretive programming environment, but mainly to the dataflow frame based approach, which makes the interpretation of multiple traces easy. The low efficiency problems revealed might be corrected by future releases of the software, and by switching to more robust commercial environments like Max/MSP.

References

[1] G. Zachmann, "Natural interaction in virtual environments," in *Workshop über Trends und Höhepunkte der Graphischen Datenverarbeitung (Proc. are English mostly)*, Universität Tübingen, Nov. 2001.

[2] N. O. Bernsen, "Natural human-human-system interaction," in *Frontiers of Human-Centred Computing, On-Line Communities and Virtual Environments*, R. E. R. Guedj, A. van Dam, and J. Vince, Eds. Springer Verlag, 2001, ch. 24, pp. 347–363.

[3] A. Valli, *Natural Interaction Homepage*. Florence, Italy: Media Integration and Communication Center, University of Florence, 2003. [Online]. Available: http://naturalinteraction.org

[4] J. Y. Han, "Low-cost multi-touch sensing through frustrated total internal reflection," in *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*. New York, NY, USA: ACM Press, 2005, pp. 115–118.

[5] P. Dietz and D. Leigh, "Diamondtouch: a multi-user touch technology," in *UIST '01, Proc. 14th annual ACM Symp. on User Interface Software and Technology*, 2001, pp. 219–226.

[6] JazzMutant^{TM}, "Lemure," 2005. [Online]. Available: http://www.jazzmutant.com/lemur_overview.php

[7] Microsoft, "Surface^{TM}," 2007. [Online]. Available: http://www.microsoft.com/surface/index.html

[8] F. Bérard, "The magic table: Computer-vision based augmentation of a whiteboard for creative meetings," in *PROCAM Workshop, IEEE Int. Conf. in Computer Vision*, 2003.

[9] M. Kaltenbrunner, S. Jordà, G. Geiger, and M. Alonso, "The reactable*: A collaborative musical instrument," in *WETICE*. IEEE Computer Society, 2006, pp. 406–411.

[10] M. Kölsch, R. Bane, T. Höllerer, and M. Turk, "Multimodal interaction with a wearable augmented reality system," *IEEE Computer Graphics and Applications*, vol. 26, no. 3, 2006.

[11] M. Lucente, "Dreamspace: Natural interaction," in *Intelligent Environments. Papers from the 1998 AAAI Spring Symposium*, ser. Technical Report SS-98-02. AAAI, March 1998.

[12] H. Ishii and N. Miyake, "Teamworkstation: Towards an open shared workspace," *Communications of the ACM*, 1991.

[13] M. Porta, "Vision-based user interfaces: methods and applications," *Int. J. Hum.-Comput. Stud*, vol. 57, no. 1, 2002.

[14] VVVV Group, "Vvvv: a multipurpose toolkit, vvvv.org," 2007. [Online]. Available: http://www.vvvv.org/

[15] Cycling '74, "New tools for media," 2008. [Online]. Available: http://www.cycling74.com/