EXTENDING AN EXISTING USER INTERFACE TOOLKIT TO SUPPORT GESTURE RECOGNITION

James A. Landay and Brad A. Myers

School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213, USA
(412) 268-3564
{landay,bam}@cs.cmu.edu

ABSTRACT
Gestures are a powerful way to specify both objects and operations with a single mark of a stylus or mouse. We have extended an existing user interface toolkit to support gestures as a standard type of interaction so that researchers can easily explore this technology.

KEYWORDS
Gesture Recognition, User Interfaces, Pen, Stylus, Toolkits, Direct Manipulation, Interaction Techniques

INTRODUCTION
With the advent of small, yet powerful portable computers, gestural input is becoming more common as an alternative to keyboard input. For example, GO's Penpoint operating system [1] is nearly entirely gesture-based. There are approximately a dozen basic gestures for navigating through and displaying documents, and many more for application-specific editing of graphics and text.

In addition to its use on small pen-based machines, gestural input with a mouse has been shown to be useful in applications targeted at desktop-based workstations. Unfortunately, most of the current user interface toolkits intended for desktop workstations (*e.g.*, Mac toolbox, X-Windows, and almost all of the toolkits built on top of either system) do not support gesture recognition. Developers and researchers who wish to experiment with this new technology must either hard-code recognizers themselves or are forced to buy entirely new systems.

In order to address this problem we have extended the Garnet User Interface Development Environment [3] to support gesture recognition. Garnet was specifically designed to ease the exploration of advanced and novel user interface techniques, so extending it to recognize gestural input was a good test of whether or not that design goal was met.

In the rest of this paper we give an overview of the usefulness of gesture recognition, describe the algorithm used for training and classification, explain how we integrated this with the Garnet Interactor Model, and finally conclude with areas for future research.

GESTURE RECOGNTION
The power of gestures comes from the ability to specify with a single mark a set of objects, an operation, and other parameters. In general, many powerful gestures can be specified with a single path, *i.e.* the pen is not lifted during the gesture. For example, Figure 1 shows single path gestures for creating a rectangle, creating a circle, copying an object, and deleting an object, respectively. These gestures are used in a simple drawing program. The 'L' gesture is the command for a rectangle to be created. The user indicates the end of the gesture by either releasing the mouse button (or if using a stylus as the input device, lifting the stylus) immediately after drawing the 'L'. The size of the 'L' is then used to determine the length and width of the rectangle, while the starting point specifies the location of the rectangle.

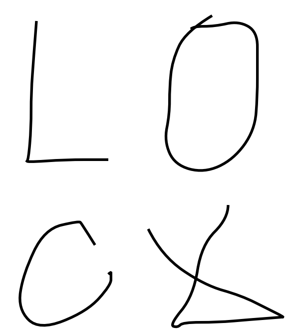

Figure 1. Rectangle, Circle, Copy, & Delete gestures

Gestures can also be useful in non-graphical applications. For instance, in a document processing program, proofreading gestures can be used to move text, delete text, change case, *etc*. By using gestures that are similar to standard proofreading marks, we can allow proofreaders and writers familar with these marks to easily perform editing operations directly on a computer using domain knowledge they already possess.

Although the above examples show that gestural input can be quite useful in direct-manipulation interfaces, developers must still overcome the problem of having to build tailored gesture recognition algorithms for each set of gestures they wish to recognize, as in [2], and the difficulty of smoothly integrating gestural input with the rest of their application interface. We have chosen to use Rubine's single-path gesture recognition algorithm [5] because it alleviates the first problem by producing gesture recognizers that are trained by example. The second problem is overcome by integrating the gesture recognition engine with the Garnet interactor mechanism.

TRAINING AND CLASSIFICATION

Rubine's classification algorithm uses statistical pattern recognition techniques in order to train classifiers and then recognize gestures. We have implemented a simple training application, Agate, which allows the developer to draw several examples (10–15) of each gesture and then save the resulting classifier.

In order to create the classifier, the algorithm calculates a set of twelve distinguishing features found in the gesture path, such as the size of the bounding rectangle of the path, the sum of the angles traversed by the path, and the angle between the starting and ending points of the gesture. It then uses statistical techniques to create a classifier based on the features extracted from the examples. In order to classify a given input gesture, the algorithm computes the distinguishing features for the gesture, and returns the best match with the learned gesture classes.

CREATING A NEW INTERACTOR

In order to simplify the use of gestural input in the interface of an application, we created a Garnet gesture interactor. The interactor model encapsulates interactive behaviors into a few "Interactor" object types [4]. Interactors are independent of the particular graphics used for the feedback and allow a clean separation of the graphics, the input handling, and the application program. Since gestural input is essentially a new kind of interaction, we created a new interactor type, the gesture interactor.

The developer creates an instance of the gesture interactor and parameterizes it according to their needs. For example, the developer specifies what event causes the interactor to start (*i.e.*, middle mouse button down or stylus down), where on the screen the mouse must be for the interactor to start, what gesture classifier to use, and what function to call with the result of the classification. Then when the user draws a gesture, the interactor uses the given classifier to determine to which gesture class the input gesture belongs. This result is then passed to the developer's callback function which can then decide what action to take depending on the result of the classification. In the simple drawing application that uses the gestures shown in Figure 1, the callback function is simply a case statement that creates new shapes, copies shapes, or deletes shapes, depending on the gesture class passed by the interactor.

In addition to passing the callback function the name of the recognized gesture, the interactor also passes it the features of the gesture. The application can use these features to parameterize the operation to be performed. For example, in the case of the 'L' gesture we use the bounding box size to determine the size of the rectangle to create.

Since gestures are a standard Garnet interactor, developers can easily create interfaces that mix traditional direct manipulation interactions with gestures. For example, a drawing program can have both a pallette from which the user can select new objects to create and a gesture interactor that allows the user to create and manipulate objects.

FUTURE RESEARCH AND CONCLUSIONS

Developing Garnet applications that use gestural input is as easy as developing applications that use menus, palettes, or any other type of input. However, there is still work to be done. If an application designer wishes a gesture to be independent of size or orientation, she must provide examples which vary in size or orientation while training the classifier. It may make more sense to explicitly declare that certain gestures are independent of size or orientation.

An additional improvement would be to allow context-dependent gestures. For example, a horizontal slash might mean to insert a minus sign if drawn in the background, to delete an object if drawn across an object, or to connect two objects if one end is in one object and the other end is in another. The system needs a way to map the same gesture into multiple meanings based on the context. This would either require the gesture interactor to decide among several different classifiers based on the context or the classification algorithm to be modified so that each classifier can use the context information.

In the future, gestural input may be as common as other standard types of interaction are today, such as menus and palettes. Future user interface toolkits should be designed with gestures as a basic input type or so that they can easily be added. Although Garnet was not designed with gestures in mind, its flexible interactor model made it easy to integrate gestures with the different forms of interaction that were already supported.

REFERENCES

[1] Robert Carr and Dan Shafer. *The Power of PenPoint.* Addison Wesley, New York, 1991.

[2] Tyson R. Henry, Scott E. Hudson, and Gary L. Newell. Integrating Gesture and Snapping into a User Interface Toolkit. In *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology*, 1990, pp. 112-122.

[3] Brad A. Myers, *et al.* Garnet: Comprehensive Support for Graphical, Highly-Interactive User Interfaces. In *IEEE Computer,* Vol. 23, No. 11, November 1990, pp. 71-85.

[4] Brad A. Myers. A New Model for Handling Input. In *ACM Transactions on Information Systems*, Vol. 8, No. 3, July 1990, pp. 289-320.

[5] Dean Rubine, Integrating Gesture Recognition and Direct Manipulation. In *Proceedings of the Summer '91 USENIX Technical Conference*, June 1991, pp. 95-100.