Gesture Avatar: A Technique for Operating Mobile User Interfaces Using Gestures

Hao Lü[*]
Computer Science and Engineering
DUB Group, University of Washington
Seattle, WA 98195
hlv@cs.washington.edu

Yang Li
Google Research
1600 Amphitheatre Parkway
Mountain View, CA 94043
yangli@acm.org

ABSTRACT

Finger-based touch input has become a major interaction modality for mobile user interfaces. However, due to the low precision of finger input, small user interface components are often difficult to acquire and operate on a mobile device. It is even harder when the user is on the go and unable to pay close attention to the interface. In this paper, we present Gesture Avatar, a novel interaction technique that allows users to operate existing arbitrary user interfaces using gestures. It leverages the visibility of graphical user interfaces and the casual interaction of gestures. Gesture Avatar can be used to enhance a range of mobile interactions. A user study we conducted showed that compared to Shift (an alternative technique for target acquisition tasks), Gesture Avatar performed at a much lower error rate on various target sizes and significantly faster on small targets (1mm). It also showed that using Gesture Avatar while walking did not significantly impact its performance, which makes it suitable for mobile uses.

Author Keywords
Touchscreens, finger-based touch input, gestures, mobile devices, target acquisition.

ACM Classification Keywords
H.5.2. Use Interfaces: Input devices and strategies, Interaction style. I.3.6. Methodology and Techniques: Interaction techniques.

INTRODUCTION

Touchscreen mobile devices have become prevalent in the recent years [2,12]. However, although finger-based touch input is intuitive, it suffers from low precision due to two fundamental problems: the area touched by the finger is much larger than a single pixel—the fat finger problem—and the pointing finger often occludes the target before touching it—the occlusion problem [21].

Although existing mobile interface widgets are often designed to be easily operated by finger, the tension remains between the low precision of finger input and the high precision needed by graphical user interfaces for the following reasons. First, it consumes precious screen real estate to make UI widgets large enough to be finger-operable, especially as the resolution of mobile devices increases, e.g., the iPhone 4's resolution is 640 × 960 [12]. Secondly, in a web interface, many UI widgets, e.g., hyperlinks or an embedded Flash player, are still small and hard to operate by finger. Lastly, high precision interactions are more challenging for mobile users because they are often on the go, e.g., walking or taking a bus, and cannot pay a close attention to the interface.

Although many solutions have been explored for addressing the fat finger and occlusion problems, most of them still require highly precise visual perception and motor control such as carefully adjusting a finger's position [1,4,18,20].

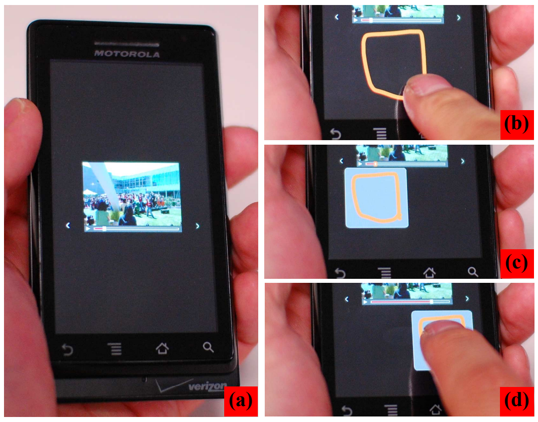


Figure 1: (a) Small mobile interface widgets are difficult to operate through fingers. (b) In Gesture Avatar, the user can draw the shape of the target, e.g., the rectangle for the knob of the slider. (c) Gesture Avatar finds the widget that best matches the user gesture and shows an avatar attached to it. (d) The user can then operate the widget (e.g., the slider knob) through the avatar, e.g., by dragging the avatar to move the video slider.

[*] This work was conducted during an internship with Google Research.

In contrast, gesture-based techniques, e.g., panning a map by swiping a finger across the touch screen, allow for casual interaction. However, gesture-based interactions tend to lack visibility compared to graphical user interfaces whose operation semantics are embodied by their interface components [16]. Prior work such as Escape [24] creatively combined the advantages of both worlds using rectilinear gestures to select graphical objects. However, it is limited in that it requires a target interface to be designed in a specific way. In addition, prior work mostly focused on the one-shot target acquisition, leaving other common interactions, such as drag-and-drop, unexplored.

To address these issues, we designed and implemented Gesture Avatar, a novel interaction technique that allows users to operate mobile user interfaces using gestures. Figure 1 provides a quick demonstration of the technique. Gesture Avatar leverages the visibility of graphical user interfaces and the casual interaction of gestures. A user can dynamically associate a gesture shape with an arbitrary GUI widget and then interact with the widget through the drawn gesture, which is conceptually akin to an avatar.

Gesture Avatar can be used to enhance a range of mobile interactions. A user study we conducted showed that in target acquisition tasks, Gesture Avatar achieved a significantly lower error rate than Shift [20] and performed faster on small targets (1mm). It also showed that walking had no significant effect on using Gesture Avatar, which makes it suitable for mobile uses. We demonstrate the feasibility and ease of use of Gesture Avatar by integrating it with existing systems such as a mobile web browser.

In the rest of the paper, we first provide a brief introduction of how a user interacts with the existing mobile interfaces through Gesture Avatar and then discuss related work. Next, we provide a detailed description of our design and implementation of Gesture Avatar along with more example applications that our technique can enhance. We then evaluate Gesture Avatar by comparing it to Shift. Finally, we discuss the factors in designing Gesture Avatar and possible future work.

MOBILE INTERACTION USING GESTURE AVATAR

In this section, we describe how a user would interact with mobile interfaces using Gesture Avatar. Assume that a user opens the New York Times website (http://nytimes.com) in a mobile phone web browser. The interaction objects such as hyperlinks are viewable but too small to be easily tapped on by a finger.

Here the user wants to navigate to the sports section. She draws an “s” on the screen[1] near the link “SPORTS” (see Figure 2a). This creates a gesture avatar, the gesture stroke with a translucent background. The bounding box of the link that the avatar is associated with is highlighted (see Figure 2b). The user can then easily trigger the link “SPORTS” by tapping on the gesture avatar, which is a much larger target.

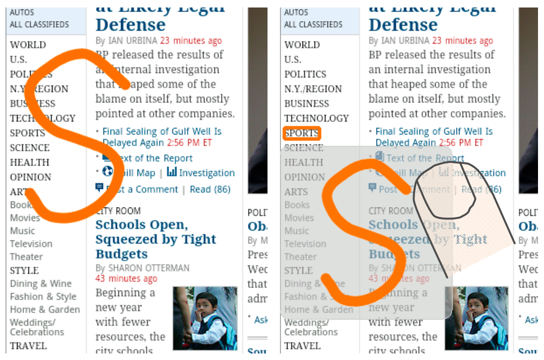


Figure 2: Tapping on a target using Gesture Avatar. (a) The user draws an “S” near the target “SPORTS”. (b) The user taps on the gesture avatar as if tapping on the target.

As we will discuss in the following sections, to form the association between a gesture and an object, we leverage both the shape of the gesture and its distance to the objects on the screen.

From this example, we can see that a gesture avatar provides a larger effective area for a user to comfortably interact with a small target. The type of interaction that Gesture Avatar can enhance is not limited to tapping. A gesture avatar essentially re-dispatches whatever touch events it receives to the associated object. For example, a user can long press an avatar to long press the associated object to bring up a context menu.

The gesture that is used to create an avatar can also be arbitrary. It can be a character or an arbitrary shape. For example, the user can draw a box to create an avatar for controlling the progress bar of a media player as seen in Figure 1. The user can then move the knob of the progress bar by dragging the avatar.

An avatar may be associated with an undesired object due to the inaccuracy of gesture recognition and the essential ambiguity of the objects on the screen. For example, in Figure 2a, when the user draws an “S”, both “SPORTS” and “SCIENCE” are good candidates.

When an avatar is associated with an undesired object, the user can either dismiss the avatar by tapping outside of the avatar and redraw a gesture, or re-assign an object to the avatar using directional (collinear) gestures. When the user draws directional gestures outside the avatar (e.g., Figure 3a), Gesture Avatar will find the next best match in the direction of the stroke and update the avatar’s position on the screen accordingly (e.g., Figure 3b). The user can repeat this process until the avatar is associated with the target.

[1] To allow the user to indicate a drawing action instead of panning the webpage, an implementation of Gesture Avatar can use various mode-switching techniques, which we will discuss later.

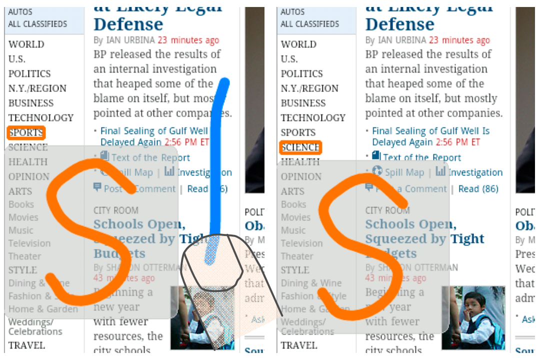


Figure 3: Re-assigning a target to the avatar. (a) The user draws a directional gesture outside the avatar. (b) Gesture Avatar finds the next best match in the given direction.

The directional gestures of Gesture Avatar enable users to easily navigate in the possible matches that are presented on the screen (a 2-dimensional space).

RELATED WORK

A large corpus of prior work has focused on facilitating target acquisition. Most of this work generally reduces the Fitts's Index of Difficulty [10] by increasing a target's size. Semantic pointing [6] dynamically adjusts the control-display (CD) ratio as the cursor approaches the target. Bubble cursor [11], which is based on area cursor [23], dynamically changes its size according to the proximity of surrounding objects so that the pointer is snapped to a nearby object.

High-precision pointing is hard with an imprecise input such as a bare finger. Many methods and techniques have been presented for high-precision touchscreen interaction. Albinsson and Zhai [1] propose two techniques using widgets for precise finger positioning by zooming, reducing CD gain, and discrete pixel-by-pixel adjustment. They show that though these techniques are faster than Offset Cursor for targets smaller than 0.8 mm, they are slower for targets larger than 3.2 mm. Benko et al. [4] present Dual Finger Selections that facilitate pixel-accurate targeting by adjusting the CD ratio with a secondary finger.

Much work has also been done in avoiding finger occlusion. Offset Cursor [18] eliminates finger occlusion by showing a cursor above users' fingers. However, users can no longer aim for the target directly even when the target is big enough. Instead, users have to first touch the surface to show the cursor and then move the cursor onto the target to select it. This contradicts the intuitiveness of direct touch. Shift [20] improves Offset Cursor by showing a copy of the occluded screen in a callout above the user's finger. Unlike Offset Cursor, users still can aim for the actual target with Shift. Their user study shows that Shift has a much lower error rate than unaided touchscreen and is faster than Offset Cursor for large targets. One problem with Shift is that the callout view is taken out of context, which requires extra effort to re-understand the view and re-recognize the target.

Wigdor et al. [22] propose back-of-device interaction to eliminate the finger occlusion problem. However, they cause the user's fingers to be occluded by the device, causing large error rates. LucidTouch [21] addresses the problem by introducing pseudo-transparency, which creates an illusion that users can see their occluded fingers, decreasing error rates. Baudisch et al. [3] propose a similar technique on small devices.

Escape [24] is a gesture-based technique for facilitating target acquisition. It first assigns directions to all objects. Users then can press their fingers near the target, followed by a directional stroke to specify the direction of the target. Through gestural interaction, Escape greatly reduces the effort in human visual feedback loop and avoids the occlusion and fat finger problems. The major limitation of Escape is that the appearance of the user interface has to be modified in order to make the assigned directions visible. Escape is demonstrated with a map-like application where using directional balloons is possible. However, it is often inappropriate or impossible to change the appearance of an application's user interface, which makes the technique less applicable. Moreover, as mentioned in their own paper, the density of the objects (the number of the objects in a certain area) is limited due to the fact that close objects cannot have similar directions, and the technique is limited near screen edges, as gestures cannot go beyond screen edges.

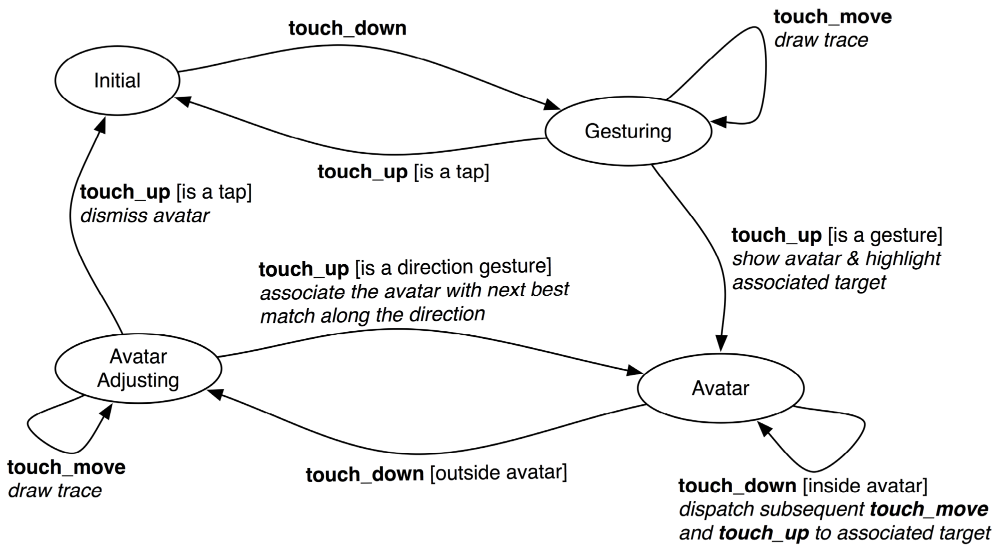
THE DESIGN OF GESTURE AVATAR

In this section, we describe the details of how Gesture Avatar works. Our design goal is to allow users to easily interact with mobile interfaces, especially when the target (e.g., a link or a button) is small and users are on the go. As a result, we designed Gesture Avatar with reducing or eliminating high precision interaction in mind.

The Interaction Flow of Gesture Avatar

Figure 4 illustrates the interaction flow of Gesture Avatar. The interaction with Gesture Avatar involves four states. The interaction process starts from the *Initial* state. Once the finger touches the screen, it enters the *Gesturing* state, in which the touch trace is rendered on top of the underlying user interface that is being operated. When the user lifts her finger and the trace is a valid gesture (e.g., the bounding box of the trace is large enough to be considered as a gesture instead of a tap), the process enters the *Avatar* state, in which the drawn gesture stroke forms the avatar with a translucent background. The object that is associated with the avatar is also highlighted.

In the *Avatar* state, the user can choose to operate the associated object through the avatar, or adjust the association if the avatar is associated with an undesired object. In this state, if the user touches down inside of the avatar, the *touch_down* as well as the subsequent *touch_move* and *touch_up* events will be re-dispatched to



touch_down
Initial
touch_move
draw trace
Gesturing
touch_up [is a tap]
touch_up [is a tap]
dismiss avatar
touch_up [is a gesture]
show avatar & highlight associated target
touch_up [is a direction gesture]
associate the avatar with next best match along the direction
Avatar Adjusting
Avatar
touch_move
draw trace
touch_down [outside avatar]
touch_down [inside avatar]
dispatch subsequent touch_move and touch_up to associated target

Figure 4: The interaction flow of Gesture Avatar. The interaction starts from the Initial state. Touch events are shown in bold. The conditions for a transition to occur are enclosed in brackets and the actions of a transition are shown in italic.

the associated object in the underneath user interface. If the user touches outside of the avatar, the user can either dismiss the shown avatar by a tap or change the associated object using directional gestures.

Matching an Gesture against Objects

In this section, we discuss how a gesture, g, is matched against the potential targets available on the screen, T, so that the avatar can be associated with a desired target, t_{match} (see Equation 1).

$$t_{match} = \underset{t_i \in T}{\arg\max} P(t_i|g) \tag{1}$$

As discussed earlier, we leverage both the semantics of the gesture and its spatial relationship with the objects on the screen. The combination of these two factors can effectively triangulate a desired object on the screen. As a result, Equation 1 can be expanded as the following:

$$P(t_i|g) = \alpha P(c_i|s_g) P(p_i|b_g) \tag{2}$$

where $P(c_i|s_g)$ encodes how much the content of the object, c_i, matches the semantics of the gesture, s_g, and $P(p_i|b_g)$ captures the spatial relationship between an object and the gesture, in particular, the distance between the gesture's bounding box, b_g, and the center of the object, p_i. We discuss how these two quantities are calculated in the following sections. (α is the normalizing constant.)

Matching Based on Semantic Information

A gesture can be a character or an arbitrary shape, and the content of an object is either a set of characters for a textual object or the contour of a graphical object (e.g., the triangular boundary of the play button of a media player). Character and shape recognition have several fundamental differences, and distinct approaches have been developed for handling each case [5]. As a result, we first classify whether the gesture is a character or a shape and then employs different recognizers for each case. For character matching, Gesture Avatar employs a neural network handwriting recognizer that recognizes letters and numbers used in English. The recognized character is then used to search against the content of each object on the screen. For shape matching, Gesture Avatar uses a template-based shape recognizer that is conceptually akin to [14]. The templates, i.e., the contours of all the objects on the screen, are added on the fly. The shape recognizer then finds the object that has the most similar contour to the gesture. Note that rather than only keeping the best guess, this calculation gives a distribution of all possible characters and shapes, which is fed into $P(c_i|s_g)$ of Equation 2.

Matching Based on Spatial Relationships

Since users tend to aim for the target, the position of the gesture for interacting with the target is often a strong indicator of where the target is. As a result, it is reasonable to assume that the closer an object is to the gesture, the more likely it is the target.

To capture this intuition, we use a 2D Gaussian distribution over the distance between the center of an object and the bounding box of the gesture. To save computation when there are numerous objects on the screen, we use the Manhattan distance, instead of the Euclidean distance, of an object to the bounding box (see Figure 5). The output of the Gaussian provides an indication of how likely the object is the target, i.e., $P(p_i|b_g)$.

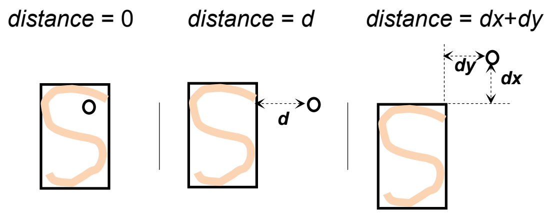


Figure 5: The distance of a target (denoted as a circle) to the avatar is calculated as the smallest Manhattan distance from it to the gesture's bounding box.

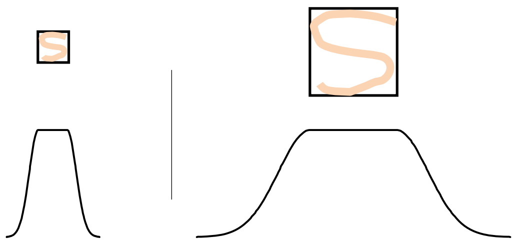


Figure 6: The larger the bounding box is, the less impact the distance has on the scoring.

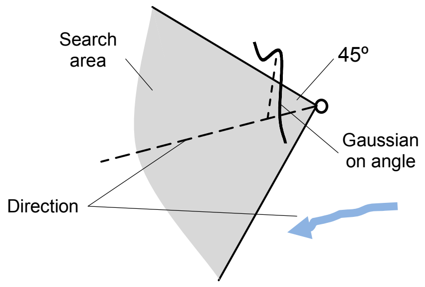


Figure 7: Searching the next best match in the given direction. We restrict the search area to the objects in a 45-degree range. A Gaussian function on the angular distance from the given direction gives a scoring factor.

The 2D Gaussian distribution employs a changing variance that is dynamically adjusted based on the length of the longer edge of the gesture's bounding box (see Figure 6). This design of our algorithm is based on the observation that users draw gestures in different sizes, and the smaller the gesture is, the more specific area the user is targeting at.

Correcting a Mismatch by Navigating in Objects

Due to the inaccuracy of gesture recognition and the inherent ambiguity of objects available on the screen, it is unavoidable that an avatar is associated with an undesired object. For example, multiple objects with similar content might be cluttered in one area. Furthermore, objects on the edges of the screen are also prone to incorrect matching, since users cannot draw gestures over these objects and the Gaussian distribution cannot efficiently capture this situation.

One typical approach to correct a mismatch is to ask the user to dismiss the avatar and then recreate it. However, this approach cannot guarantee success and can be annoying if it is only a close miss, e.g., when the target object is just next to the mismatched object.

In our design, we allow the user to go to the next best match in the given direction by drawing a directional stroke. Directional strokes are fast to perform and the outcomes of them are more predictable.

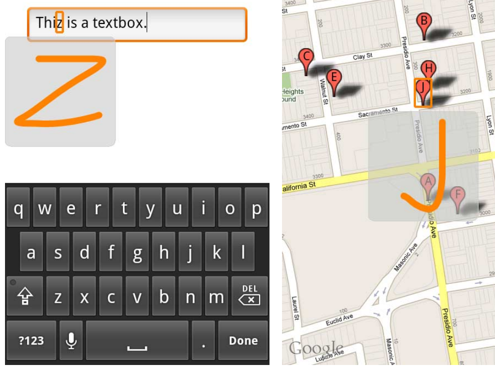
Based on the directional stroke the user draws, we infer the next best match based on not only how well an object matches the avatar gesture semantically, but also the angular distance between the directional stroke and the direction from the currently matched object to the object being evaluated. Again, we use a Gaussian function to give each object a score based on its angular distance (see Figure 7). To reduce computation cost, we restrict the search to the objects within ±45º of the stroke direction.

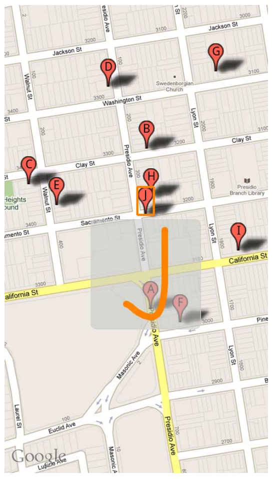
MORE EXAMPLES

In the previous sections, we demonstrated the use of Gesture Avatar through a media player and a mobile browser. In this section, we discuss two more examples.

The first example (see Figure 8a) applies Gesture Avatar to moving the caret in a text box. The on-screen keyboards on most touchscreen mobile phones do not have dedicated keys for moving the caret. Instead, users need to directly touch between two characters. Since the text is small, moving the caret is error-prone. The iPhone uses a Shift-like technique to address this problem. With Gesture Avatar, the user can draw the character before (or after) the desired position to assign an avatar to it and then tap on the right (or left) half of the avatar to move the caret.

The second example (see Figure 8b) applies Gesture Avatar to Google Maps. Previously, maps have been a heavily used example to demonstrate and evaluate different target acquisition techniques [20,24]. Locations on a map are typically represented by distinct letters. This property makes Gesture Avatar promising for acquiring locations,





(a) **(b)**

Figure 8: (a) Moving the caret in a text box using Gesture Avatar. (b) Acquiring a location in Google maps.

since there is little semantic ambiguity in these location objects.

IMPLEMENTATION

Gesture Avatar was primarily written in Java using Android SDK 2.2. For image contents, we first compute their edges using the Canny edge detector and then pass them to our shape recognizer. The image processing and edge detection were written in C++ using OpenCV [17] and built with a variation of Android NDK.

The example applications that we have discussed in the paper were written in Java. We built these examples by wrapping an existing user interface, e.g., a web browser or a text input field, with an additional interaction layer, i.e., Gesture Avatar. In the mobile web browser example, the current version of mobile WebKit does not expose its UI structures, i.e., the DOM tree, in Java but only accessible through JavaScript. As a workaround, we implemented the logic for searching the best matches in JavaScript and inject the JavaScript code upon loading a webpage.

EVALUATION

As discussed earlier, Gesture Avatar addresses not only the acquisition of small targets, but also the follow-up interaction such as dragging an object after it is acquired. There has been little prior work in addressing the latter. So Gesture Avatar makes a clear contribution in that regard. As a result, we here focus on its performance on target acquisition compared to Shift.

We hypothesize that Gesture Avatar outperforms Shift on small targets and is better for mobile uses. Specifically:

(H1) Gesture Avatar will be slower than Shift on larger targets, but faster on small targets.

(H2) Gesture Avatar will have fewer errors than Shift.

(H3) Mobile situations such as walking will decrease the time performance and increase the error rate of Shift, but have little influence on Gesture Avatar.

Apparatus

The experiment was conducted on a Motorola Droid running Android 2.2 (see Figure 1). It has a 3.7-inch, 480×854 pixel multi-touch display. The walking tasks were conducted on a treadmill desk from Steelcase, with a maximum speed of 2.0 mph (see Figure 9b). Walking on a treadmill offers a reasonable simulation that captures common properties of various "on the go" scenarios, such as device shaking and significant body movement. At the same time, an indoor treadmill setting allowed us to conduct the experiment in a more controlled manner.

Participants

Twelve participants (8 male and 4 female), between age 20 and 30, were recruited from a company. They were all right-handed. All the participants owned touchscreen mobile devices and used their phones extensively. Each

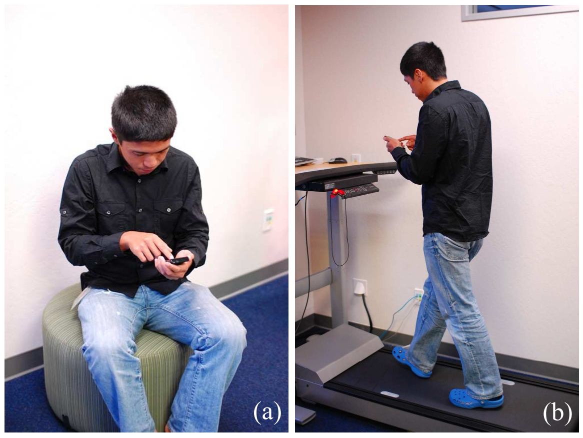


Figure 9: (a) The user was sitting on a stool. (b) The user was walking on a treadmill.

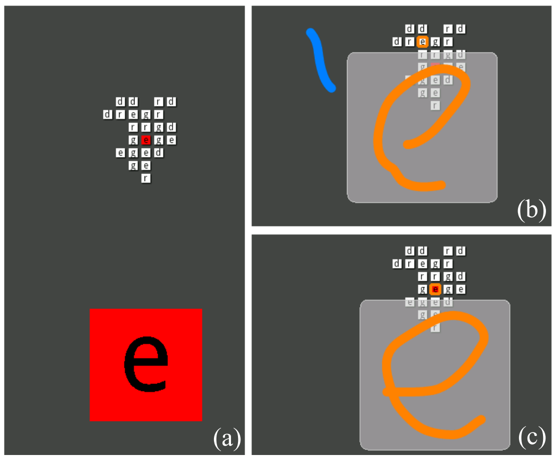


Figure 10: The experimental task. (a) The target is highlighted in red and a magnified copy serves as the start button. (b) and (c) show a typical run of the task.

participant received a gratuity equal to a US$30 gift card for their participation.

Experimental Design

A within-subjects factorial user study was used in this experiment. Half of the participants learned and performed Gesture Avatar first and then Shift, while the other half did the opposite so that the order of two techniques was counterbalanced. For each technique, participants were asked to complete the tasks in two conditions: sitting and walking as shown in Figure 9.

The objects in the tasks were represented as letter boxes (see Figure 10). Participants were asked to acquire targets of different sizes, positions, letters, and ambiguity using both Gesture Avatar and Shift. In the starting screen of a task (Figure 10a), the participants were shown 24 small letter boxes. The target box, which was always near the center, was highlighted in red. To eliminate the time for searching for the target visually, which was not the focus of this experiment, a magnified version of the target was shown 300 pixels below the target. Participants needed to tap the magnified target to start the task. Performance times were measured between the tap on the magnified target and the selection of the target.

We chose 24 letter boxes for our study for several reasons. First, we wanted to cover the entire spectrum of semantic ambiguity in targets, from very ambiguous (i.e., few letters are used with high repetitions) to non-ambiguous cases (i.e., each box has a unique letter). As a result, the number of letter boxes should not be larger than 26, the size of the English alphabet. Second, we wanted the number of unique letters shown on the screen to be a factor of the total number of boxes so that the distribution of the letter usage is uniform, i.e., each letter can have an equal number of instances and affect the gesture recognizer equally. 24 is a good option as it has many nontrivial factors (2, 3, 4, 6, 8, and 12). Lastly, 24 objects offer an information density that reasonably simulates real life scenarios.

The ambiguity of the target, i.e., the number of objects that have the same letter near the target, may affect the performance of Gesture Avatar. We simulated the ambiguity by controlling the distance between objects and the number of letters used. Since the performance of Shift correlates to the size of the objects instead of the distance between them, we fixed the margin of objects to 5px and varied the size of the objects to affect both techniques. Three sizes were used in the study: 10px (1mm), 15px (1.5mm) and 20px (2mm).

Some objects at a small size such as 1mm would become uncomfortable or difficult for users to read. However, many UI widgets in a web interface are often presented at such a scale. In addition, mobile users are often on the go. The short attention span of a mobile user as well as interacting in motion affects the performance of human motor control. These factors can greatly decrease the effective size of a target, such that a large target in such circumstances might have a much smaller effective size.

When using Gesture Avatar to acquire the target, we highlighted the currently matched object with an orange border (see Figure 10b). Participants could tap on the avatar to acquire the matched object. For this comparison, we re-implemented Shift and used the escalation time 0, 50, and 200 milliseconds for 10, 15, and 20 pixel objects, respectively.

To make the finger position more stable, we averaged the finger positions with a smoothing factor of 0.25 as a low pass filter for the finger position signals.

While performing these techniques, participants were required to acquire targets with the index finger of their dominant hands while holding the device in their non-dominant hands, as required in previous studies.

Experimental Procedure

Upon being introduced to a new technique, participants had a practice session where they performed the same type of tasks as the test sessions. They were allowed to practice until they felt comfortable using it. During the test period, participants could take a break between tasks. For each technique, participants performed the first 12 test sessions while sitting on a stool (see Figure 9a) and the second 12 while walking on a treadmill (see Figure 9b). The walking speed was between 1.8 and 2 mph depending on participants' preference. The 12 sessions covered all combinations of box sizes and numbers of unique letters. In each session, there were 10 tasks, with the same box size and same number of unique letters. They were drawn randomly from a pre-generated task pool (50 tasks for each combination).

Independent Variables

The independent variables were *Technique* (Gesture Avatar versus Shift), *MobileState* (sitting versus walking), *TargetSize* (10, 15, or 20px), and *NumOfLetters* (the number of unique letters used: 1, 4, 8, and 24). In summary, the experimental design was:

12 *Users* ×
2 *Techniques* (*Gesture Avatar* versus *Shift*) ×
2 *MobileState* (*Sitting* versus *Walking*) ×
3 *TargetSizes* (5, 15, 20px) ×
4 *NumOfLetters* (1, 4, 8, 24) ×
10 *Target Acquiring Tasks*
= 5760 tasks

Results

Here we discuss our findings, including time performance, accuracy and subjective preferences.

Time Performance

For performance time, we performed a within-subjects analysis of variance (ANOVA) for *Technique* × *MobileState* × *TargetSize* × *NumOfLetters*. A main effect was found for *Technique* ($F_{1,5758}$=51.574, p<0.001), *MobileState* ($F_{1,5758}$=18.668, p<0.001), and *TargetSize* ($F_{2,5757}$=140.736, p<0.001), but not *NumOfLetters*. The significant interactions were *Technique* × *MobileState* ($F_{1,5756}$=19.070, p<0.001), *Technique* × *TargetSize* ($F_{2,5754}$=122.165, p<0.001), *Technique* × *NumOfLetters* ($F_{3,5752}$=5.170, p<0.001), *MobileState* × *TargetSize* ($F_{2,5754}$=5.760, p<0.005) and *Technique* × *MobileState* × *TargetSize* ($F_{2,5748}$=9.583, p<0.001).

Tukey's post-hoc pair-wise comparison showed that Gesture Avatar was significantly slower than Shift when *TargetSize* was 20px, but significantly faster than Shift when *TargetSize* was 10px. There was no significant difference between two techniques when *TargetSize* was 15px. Both techniques were significantly faster on larger target sizes. Shift with *MobileState* as sitting was significantly faster than it with *MobileState* as walking. For Gesture Avatar, there was no significant difference between *MobileState* being sitting versus walking.

We performed the same analysis on Shift and Gesture Avatar separately. We found that for Gesture Avatar, there was no significant difference across different *MobileStates* and *TargetSizes*. For Shift, *TargetSize*=10px was significantly slower than larger sizes at all *MobileStates*, and sitting was significant faster than walking when the *TargetSize* is 10px.

Error Rates

For the error rate, we also performed a within-subjects ANOVA for *Technique* × *MobileState* × *TargetSize* × *NumOfLetters*. A main effect was found for *Technique* ($F_{1,5758}$=674.166, p<0.001), *MobileState* ($F_{1,5758}$=26.493, p<0.001), and *TargetSize* ($F_{2,5757}$=41.057, p<0.001), but not *NumOfLetters*. The significant interactions were *Technique* × *MobileState* ($F_{1,5756}$=18.878, p<0.001), *Technique* × *TargetSize* ($F_{2,5754}$=41.201, p<0.001), and *MobileState* × *TargetSize* ($F_{2,5754}$=6.729, p<0.001).

Tukey's post-hoc pair-wise comparison showed that Gesture Avatar had lower error rates than Shift on all *TargetSizes*, while there was no significant difference for Gesture Avatar across different *TargetSizes*.

We performed the same analysis on Shift and Gesture separately. We found that for Gesture Avatar, there was no significant difference across all *MobileStates* and *TargetSizes*. For Shift, *TargetSize*=10px when walking had a significantly higher error rate than any other *TargetSizes* and *MobileStates*.

Subjective Preferences

In the post-study questionnaire participants were asked about their preferences for the two techniques. 10 of the 12 participants preferred Gesture Avatar, 1 Shift, and 1 both. All commented that Shift was more stressful especially while walking. 10 participants agreed that Gesture Avatar is useful, 1 strongly agreed and 1 neutral. 8 participants agreed that Gesture Avatar is easy to use, 3 strongly agreed and 1 neutral.

Experiment Discussion

Our results support hypothesis H1. Figure 11 and Figure 12 show that Shift's performance time increases as the target size decreases. However, Gesture Avatar's performance time almost remains unaffected by the change of the target size. As a result, Gesture Avatar is slightly slower than Shift when the target size is 20px, but quickly catches up and becomes a lot faster when it is 10px. This also implies that Gesture Avatar scales well to small targets.

Our results support hypothesis H2. Figure 13 and Figure 14 show trends in error rates that are similar to those of performance time. However, Gesture Avatar has much lower error rates than Shift. Though Gesture Avatar has a correction stage, achieving such low error rates with little sacrifice in performance is still surprising.

Our results also support hypothesis H3. By comparing Figure 11 to Figure 12, and Figure 13 to Figure 14, we can

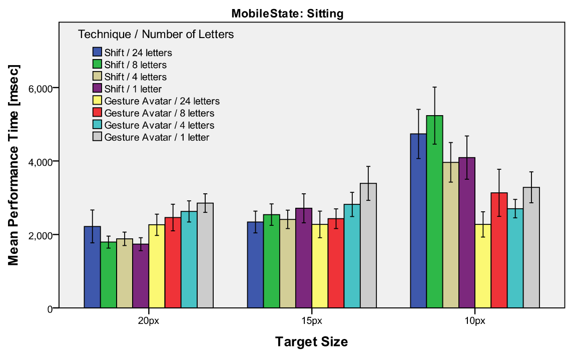


Figure 11: Mean performance time for Technique × TargetSize × NumOfLetters when participants were sitting on a stool. In this and all later charts, error bars represent 95% confidence intervals.

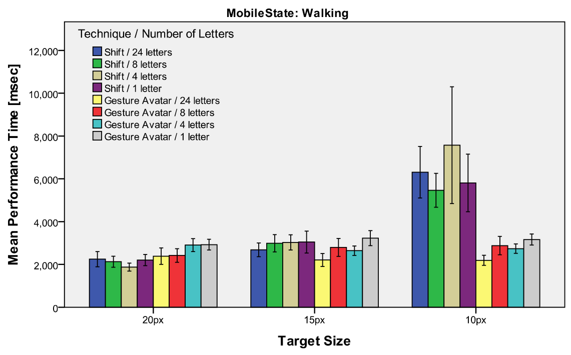


Figure 12: Mean performance time for Technique × TargetSize × NumOfLetters when participants were walking on a treadmill.

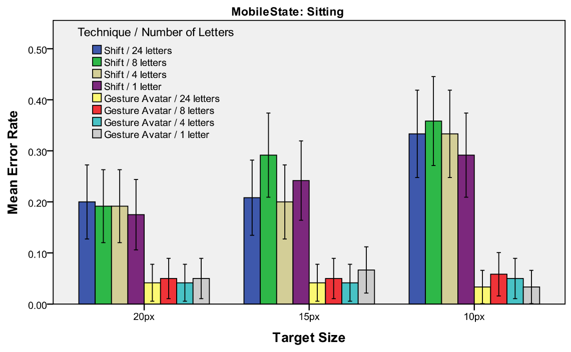


Figure 13: Mean error rate for Technique × TargetSize × NumOfLetters when participants were sitting on a stool.

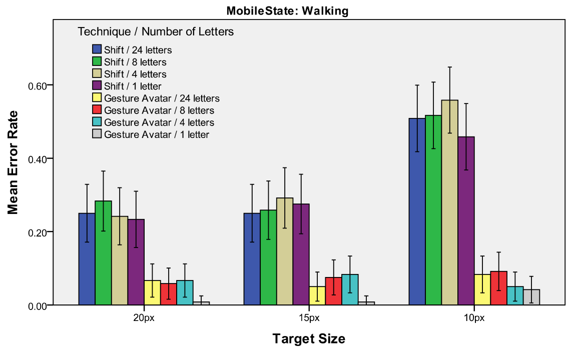


Figure 14: Mean error rate for Technique × TargetSize × NumOfLetters when participants were walking on a treadmill.

see that walking greatly hurt Shift both in performance times and error rates, but had almost no influence on Gesture Avatar. This indicates that Gesture Avatar is promising when users are on the go.

One surprising finding is that the number of unique letters had no significant influence on Gesture Avatar, as we thought that more content ambiguity would decrease the effectiveness of the gesture recognizer, and therefore decrease the overall performance. However, in our study, we observed that the participants tended to draw smaller gestures when there was more ambiguity and as a result the matched objects were often either the targets or very close to the targets. We also observed that all participants quickly converged to "flicking" [9] towards the targets as their correction strokes, which was very efficient.

DISCUSSION

Accelerating for One-Shot Interactions

As we have discussed in the previous sections, Gesture Avatar offers unique advantages for manipulating small targets beyond target acquisition. As a result, it can address many common interaction behaviors that prior techniques cannot support, e.g., adjusting a slider, drag-and-drop, double tapping, and long pressing.

However, for a target that only accepts one-shot interaction, e.g., the only valid interaction for a button is to be clicked, it might be unnecessary to show the avatar when the match is highly unambiguous. For example, a user can simply draw a gesture near a target (such as a hyperlink) and if the ambiguity is low, the target should be triggered immediately, instead of having to display an avatar for the user's confirmation. In such case, Gesture Avatar supports a one-shot interaction as prior techniques did, although Gesture Avatar still has the advantage of allowing casual interactions.

Another design alternative is to show the avatar at the end of gesturing, but before the user lifts her finger, which gives the user an opportunity to confirm the match. If the match is incorrect, the user can either continue gesturing, or dwell the finger, e.g., for a half second, on the touch screen to bring up the avatar. She can then adjust the matching using directional gestures. This design would require us to detect the potential ending of a gesture stroke being drawn.

Acquiring Moving Targets

Prior work on target acquisition often assumes fixed locations for targets, which is not always the case. As we have seen in the previous media player example (Figure 1), the knob of the progress bar moves towards the right end while the media player is playing. As another example, in a real-time location information application, objects such as cars can move quickly on a map.

Moving targets make a technique that requires high precision difficult to use since acquiring these targets require intensive visual tracking and more challenging motor control. As a result, many existing techniques are inefficient in acquiring moving targets. In Shift, the target can move out of the callout view during the interaction, and in Escape, the target can move to items with the same or similar directions. In contrast, for Gesture Avatar, moving targets are not a challenge. Since the target matching algorithm works on a static state of the user interface, all the claims in this paper should still hold for moving targets. We do need to update the avatar view constantly in order to attach the avatar to the moving targets.

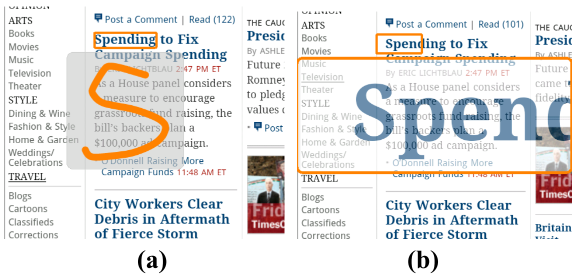


Figure 15: Two different representations we explored for Gesture Avatar: (a) the stroke and (b) the lens representation.

Avatar Representations

We explored various representations for gesture avatars. We focused on two representations in our iterative design process: *stroke* and *lens* representations (see Figure 15). We employ the stroke representation in the current design, i.e., an avatar is shown as the gesture stroke with a translucent background. The lens representation is to show a magnified version of the matched target and has been explored in literature [19]. Both representations highlight the bounding box of the original target. We chose the stroke over the lens representation for the following reasons.

First, a lens often affords different interaction behaviors that are inconsistent with the avatar metaphor that we intended to support. For example, dragging lens would be naturally interpreted as shifting the view rather than moving the magnified object. Secondly, a magnified view often has a more complicated appearance than a gesture stroke. As a result, the lens representation is more likely to obscure the underlying user interface and adds visual complexity. In contrast, the stroke representation is more lightweight.

Mode Switching Issues

Similar to many other gesture-based techniques, mode switching [15] is also an issue in integrating Gesture Avatar into existing user interfaces. For example, when gesturing on a map, it is necessary to distinguish the initial gesturing for creating an avatar from actions such as panning.

In our implementation of Gesture Avatar, a user gestures by first pressing and holding a holding finger, and then drawing gestures with a different finger, the gesturing finger. If the holding finger moves, it is considered as a regular touch-based interaction, e.g., panning or pinching to zoom. Similar asymmetric dependent two-handed techniques have been studied in [13].

Integrating with Existing Systems

Gesture Avatar needs to be aware of the objects available on the user interface being operated, although it does not require these objects to be in a specific presentation. The more Gesture Avatar knows about the underlying user interface, the better it can perform. The ideal situation is that we can access the structure of the underlying user interface. For example, the structure of a webpage can be accessed via its DOM tree [8]. However, this information can be inaccessible to Gesture Avatar (e.g., in a Flash application embedded in a webpage). One possible solution is to reverse engineer the underlying user interface from its pixels as previously explored in Prefab [7]. The limitation of this kind of approach is that it only works for known UI widgets. To make Gesture Avatar a general technique and avoid the effort for implementing it for every user interface, we designed Gesture Avatar as an interaction layer that can be easily used to wrap around a specific interface via a simple API. We implemented a toolkit based on the Android platform and all the examples demonstrated in the paper were implemented using the toolkit.

CONCLUSION AND FUTURE WORK

This paper presents Gesture Avatar, a gesture-based interaction technique for operating mobile interface that supports casual interaction of the existing mobile interfaces. We discussed the interaction design of Gesture Avatar. We showed how Gesture Avatar can improve user experience on mobile devices via four example applications. A controlled user study and the range of interactions that are supported indicate that Gesture Avatar outperforms its peers in many ways.

REFERENCES

1. Albinsson, P.-A. and Zhai, S. High precision touch screen interaction. *Proc. CHI 2003*, ACM Press (2003), 105-112.
2. Android (operating system). http://en.wikipedia.org/wiki/Android_(operating_system).
3. Baudisch, P. and Chu, G. Back-of-device interaction allows creating very small touch devices. *Proc. CHI 2009*, ACM Press (2009), 1923-1932.
4. Benko, H., Wilson, A.D., and Baudisch, P. Precise selection techniques for multi-touch screens. *Proc. CHI 2006*, ACM Press (2006), 1263-1272.
5. Bishop, C.M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. 2006.
6. Blanch, R., Guiard, Y., and Beaudouin-Lafon, M. Semantic pointing: improving target acquisition with control-display ratio adaptation. *Proc. CHI 2004*, ACM Press (2004), 519-526.
7. Dixon, M. and Fogarty, J. Prefab: implementing advanced behaviors using pixel-based reverse engineering of interface structure. *Proc. CHI 2010*, ACM Press (2010), 1525-1534.
8. Document Object Model. http://en.wikipedia.org/wiki/Document_Object_Model.
9. Dulberg, M.S., Amant, R.S., and Zettlemoyer, L.S. An Imprecise Mouse Gesture for the Fast Activation of Controls. *INTERACT*, (1999), 375-382.
10. Fitts, P.M. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of experimental psychology 47*, 6 (1954), 381-391.
11. Grossman, T. and Balakrishnan, R. The bubble cursor: enhancing target acquisition by dynamic resizing of the cursor's activation area. *Proc. CHI 2005*, ACM Press (2005), 281-290.
12. iPhone 4. http://en.wikipedia.org/wiki/IPhone_4.
13. Kabbash, P., Buxton, W., and Sellen, A. Two-handed input in a compound task. *Proc. CHI 1994*, ACM Press (1994), 417-423.
14. Kara, L.B. and Stahovich, T.F. An Image-Based Trainable Symbol Recognizer for Sketch-Based Interfaces. *AAAI Fall Symposium Series 2004: Making Pen-Based Interaction Intelligent and Natural*, (2004).
15. Li, Y., Hinckley, K., Guan, Z., and Landay, J.A. Experimental analysis of mode switching techniques in pen-based user interfaces. *Proc. CHI 2005*, ACM Press (2005), 461-470.
16. Norman, D.A. The way I see it: natural user interfaces are not natural. *Interactions 17*, 3 (2010), 6-10.
17. OpenCV. http://opencv.willowgarage.com/wiki/.
18. Potter, R.L., Weldon, L.J., and Shneiderman, B. Improving the accuracy of touch screens: an experimental evaluation of three strategies. *Proc. CHI 1988*, ACM Press (1988), 27-32.
19. Ramos, G., Cockburn, A., Balakrishnan, R., and Beaudouin-Lafon, M. Pointing Lenses : Facilitating Stylus Input through Visual- and Motor-Space Magnification. *Proc. CHI 2007*, ACM Press (2007), 757-766.
20. Vogel, D. and Baudisch, P. Shift: a technique for operating pen-based interfaces using touch. *Proc. CHI 2007*, ACM Press (2007), 657-666.
21. Wigdor, D., Forlines, C., Baudisch, P., Barnwell, J., and Shen, C. LucidTouch : a see-through mobile device. *Proc. UIST 2007*, ACM Press (2007), 269-278.
22. Wigdor, D., Leigh, D., Forlines, C., et al. Under the table interaction. *Proc. UIST 2006*, ACM Press (2006), 259-268.
23. Worden, A., Walker, N., Bharat, K., and Hudson, S. Making computers easier for older adults to use: area cursors and sticky icons. *Proc. CHI 1997*, ACM Press (1997), 266-271.
24. Yatani, K., Partridge, K., Bern, M., and Newman, M.W. Escape: a target selection technique using visually-cued gestures. *Proc. CHI 2008*, ACM Press (2008), 285-294.