ARPEGE : DESIGN AND LEARNING OF MULTI-FINGER CHORD GESTURES

BAU O / GHOMI E / MACKAY W

Unité Mixte de Recherche 8623
CNRS-Université Paris Sud – LRI

02/2010

Rapport de Recherche N° 1533

CNRS – Université de Paris Sud
Centre d'Orsay
LABORATOIRE DE RECHERCHE EN INFORMATIQUE
Bâtiment 490
91405 ORSAY Cedex (France)

Arpege: Design and Learning of multi-finger chord gestures

OLIVIER BAU, EMILIEN GHOMI and WENDY MACKAY
InSitu, INRIA/LRI

This article proposes simple guidelines for generating multi-finger chord gesture sets that specifically account for physiological and motor characteristics of the human hand. We illustrate these guidelines with a sample gesture set. Our first study shows that multi-touch chord gestures can be significantly faster than pointing on a tool palette, but they also result in significantly more errors for novice users. We then describe *Arpege*, a contextual feedforward/feedback technique that makes it easy to learn multi-touch chord gestures. In a second study, *Arpege* took no more time than a 'cheat sheet' but resulted in almost three times fewer errors. Techniques such as *Arpege* make it possible to include abstract multi-finger chord gestures in the design space, offering a rich and powerful form of interaction on a multi-touch surface.

Categories and Subject Descriptors: H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Evaluation/methodology ; Input devices and strategies ; Interaction styles ; Training, help, and documentation ; User-centered design*; I.3.6 [**Computer Graphics**]: Methodology and Techniques—*Ergonomics ; Interaction techniques*

General Terms: Design, Experimentation, Human Factors, Theory

Additional Key Words and Phrases: Arpeggios, Chords, Gestures, Feedback, Feedforward, Learning, Multitouch gestures, Multi-touch tables

1. INTRODUCTION

Multitouch surfaces, ranging from small iPhones to large table tops, offer rich new possibilities for interaction. The combination of a display and a touch-sensitive surface offers users true direct manipulation, e.g., moving an object simply by pointing to it and dragging. Many touch-sensitive surfaces can now detect multiple points of contact, e.g. [Han 2005], enabling multi-finger interaction, such as resizing a photograph by dragging the corners with two fingers.

While such technologies are becoming more prevalent, the range of actual multi-touch gestures remains limited. The problem is not recognition, per se, since identifying multiple points of contact is well understood. Rather, we lack both clear guidelines for creating multi-touch gestures and effective means for learning them.

Several researchers have addressed the first issue by examining how people actually point to and manipulate objects in the real world, with the goal of creating "natural" styles of interaction. For example, Wilson et al. [2008] developed techniques for manipulating virtual physical objects on a multi-touch surface. Wobbrock et al. [2009] proposed a set of gestures designed "by first portraying the effect of a gesture and then asking users to perform its cause".

Although these strategies help to design intuitive interfaces for novices, they offer a limited range of gesture expressivity and do not take full advantage of the capabilities of the human hand. Other domains design specialized instruments specifically for experts, providing them with a rich vocabulary for expressing a variety of commands. For example, pianists learn to play combinations of notes together, as chords or arpeggios, and stenographers map combinations of keys to syllables or phonemes [Doo et al. 2006; Szentgyorgyi and Lank 2007].

Of course, this introduces the problem of learning. How do we help the user move from novice to expert performance? Users must not only learn which commands are possible, i.e. the available vocabulary, but also how best to perform the gestures themselves, i.e. what the system can recognize. In-place help techniques, such as *Marking Menus* [Kurtenbach 1993] and *Octopocus* [Bau and Mackay 2008] provide a combination of *feedforward*, to help users learn the current set of possible commands, and *feedback*, to let users know whether or not their gestures have been successfully recognized. However, these strategies have not yet been applied to multi-touch interfaces, particularly those with arbitrary commands.

This paper begins with a review of related work. Next, we provide simple guidelines for generating multi-finger chord gesture sets that specifically account for physiological and motor characteristics of the human hand. We then illustrate these guidelines with a sample multi-finger chord gesture-set. Next, we present the results of a study that compares performance on simple pointing and multi-finger chord gestures. We address the problem of learning chord gestures by comparing on-line "cheat sheets" to *Arpege*, our multi-touch feedforward/feedback technique. We conclude with a discussion and directions for future research.

2. RELATED WORK

Several researchers have explored the use of multiple finger positions to express commands. For example, Charade [Baudel and Beaudouin-Lafon 1993] explored how to express abstract commands for controlling a slide show, using different positions of the fingers of the hand. They introduced the concept of natural and "tense" hand positions, and developed a gesture vocabulary that differentiated between unconscious gestures and explicit commands. Malik et al. [2006] used multi-finger gestures to invoke commands at a distance. Their basic vocabulary included the opening or closing of the hand, extension of certain fingers, translation of the hand and pinching.

Although these projects demonstrate the benefits of creating vocabularies with different finger combinations, they express commands "in the air". We are interested in the problem of how to express multi-finger gestures on a flat, multi-touch surface. For example, Wu et al. [2003] introduced multi-finger and whole hand gestures that leverage and extend the types of actions that people perform when interacting on physical tabletops. Users can, for example, use two "L" gestures to define their own private area. They also introduced hierarchical multi-finger interaction, e.g. their FurniturePalette acts as a two-step hierarchical menu.

Although "natural interaction", which includes direct manipulation, remains the dominant paradigm for touch-based interaction research, we argue that it makes sense to also consider how to construct and learn *abstract* multi-finger gestures. We know from other domains that people can attain extremely rapid and accurate performance on chorded instruments, such as the piano, chorded keyboards and stenotypes. For example, an expert stenographer maps phonemes from natural speech to multi-finger chords and types at an average speed of 225 English words per minute [Doo et al. 2006; Szentgyorgyi and Lank 2007].

As early as the 1960's, Engelbart [1962] explored using five-finger chords for expert computer users. The device consisted of a five-key pad designed to be used with the non-dominant hand, while the dominant hand controlled a mouse or keyboard. More recently, PreSense [Rekimoto et al. 2003] enabled users to invoke commands by creating contact patterns on a touch-sensitive keypad. Fingerworks [FingerWorks 2001] developed multi-touch surfaces that enabled users to execute multi-finger chord gestures as command shortcuts. For example, users of their *igesture* tablet could invoke a copy command by simultaneously tapping the thumb and middle finger. Apple has recently introduced dynamic multi-finger chord gestures, such as touching the surface with four fingers and sliding upward to invoke the Exposé command.

Although using multiple fingers to invoke commands has a number of potential benefits, we must ensure that they are comfortable to execute and not difficult to learn. Our first goal then is to provide gesture design guidelines that both maximize comfort and expressivity by taking the physical characteristics of the human hand into account. Few studies of expert performance exist beyond Matejka et al.'s [2009] evaluation of a chord-based mouse emulation technique. Thus, our second challenge is to develop empirical methods for testing multi-gesture chord gestures and how easily users can use them to execute commands.

Even if we can demonstrate that experts can perform efficiently and effectively with chord gestures, we still face the problem of how to help users move from novice to expert performance. As Rekimoto et al. [2003] pointed out when designing PreSense, users face a steep learning curve and need effective guidance to master the various key combinations.

Several techniques have been developed for teaching multi-touch gestures. The Multitouch Menu [Bailly et al. 2008], an extension of the multi-finger menu [Wu and Balakrishnan 2003], is invoked when the palm touches the table. Novice users can select an item from a pie menu with their thumb and see sub-menu items and their corresponding commands displayed near each finger. Expert users can invoke a command without displaying the menu by touching the thumb and a finger, providing a smooth transition from novice to expert behavior, similar to marking menus [Kurtenbach 1993].

TouchGhost [Vanacken et al. 2008] offers another approach, in which a two-handed character simulates a gesture, such as dragging two hands apart to rescale a picture, and displays the result. This offers a dynamic illustration, but requires users to watch and interpret the animation. Several commercial systems offer off-line learning techniques, such as Westerman's gesture-learning application [2008] and Elias et al.'s gesture dictionaries [2007]. FingerWorks [FingerWorks 2001] offers "cheat sheets" that provide a visual summary of all chord gestures. Shadowguides [Freeman et al. 2009] is designed to help users execute a range of multifinger and whole hand gestures by combining a cheat sheet and interactive guidance for dynamic gestures. ShadowGuides includes a "registration pose guide" for guiding static gestures, based on a variation of commercially available cheat sheets (e.g. [FingerWorks 2001]). These highlight the parts of the hands that are involved in the gestures as well as the "the user shadow expected by the system", but do not inform the user just *how* to perform the static gesture.

We are interested in a specific subset of gestures i.e. *chords* and address the problem of how to help users both learn and execute them. We also want to provide a smooth transition between novice and expert performance by including help in the context of the execution of the gesture, rather than as a geographically separated guide.

Gesture learning techniques have also been developed for the desktop, including learning hotkeys [Grossman et al. 2007] and mouse gestures [Appert and Zhai 2009]. OctoPocus [Bau and Mackay 2008] offers an alternative to static off-line help techniques. This dynamic guide helps users execute and remember commands *during input.* It provides *feedforward* to show the user the remaining available gestures and *feedback* to show how the recognition system is interpreting the input, mid-gesture. Our third goal is to modify this approach to aid learning of chord gestures on a multi-touch surface.

3. CHORD GESTURE DESIGN AND RECOGNITION

We define a *multi-finger chord gesture* as the simultaneous placement of two or more fingertips on a touch surface. Other gestures, such as path-based or whole-hand gestures, e.g. [Wu and Balakrishnan 2003], are beyond the scope of this paper. We begin with the problem of how to design a set of chord gestures from the user's perspective, independent of the problem of recognition. We need to minimize fatigue and discomfort, avoiding awkward configurations that users find difficult to learn and execute. To accomplish this, we must understand the basic physiology of the human hand and the constraints imposed by the relationships among the fingers.

3.1 Chord Gesture Design

3.1.1 *Finger placement.* For each chord gesture, we must consider which fingers are involved and their relative positions. Fingers have different motor capabilities and are more or less able to move independently of each other. Some fingers are weak, such as the pinky, others are strong, such as the index finger. Some move freely, such as the thumb, whereas others are highly dependent upon neighboring fingers, such as the ring finger. Note that some users, such as experienced pianists, have highly trained fingers and are likely to be able to execute a wider range of chord gestures. However, our design guidelines are intended for average users. We propose two basic rules, based on the physiology of the human hand:

1: *Exclude chord gestures that require lifting either the middle or ring finger without also lifting at least one of their neighbors.*

We first consider the "natural" position of a hand, i.e. the palm is in contact with the surface, as well as the side of the thumb and the tips of each finger, all slightly curved (fig.1-a). From this position, the user can independently raise or lower each finger, implying that each hand has $2^5 - 1 = 31$ possible finger-lift combinations. If we exclude the five one-finger configurations, 26 combinations remain. We next account for the fact that fingers do not move independently of each other. Lee et al. [1995] measured the interdependence of each finger, excluding the thumb, and

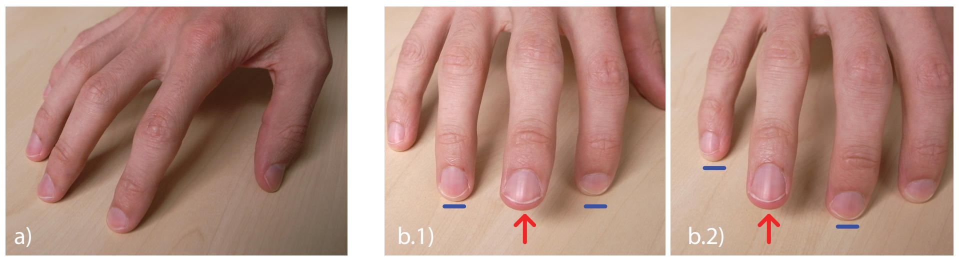
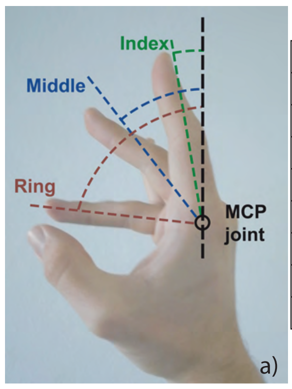


Fig. 1. a) Natural hand placement on a flat surface. b) Lifting the middle and ring fingers is awkward.

specified the angular constraints of the Metacarpophalangeal joints (fig.2). They then determined the flexion limits for each finger, as imposed by the neighboring fingers, and quantified how easy or hard it is to lift each finger independently.



	Index	Middle	Ring	Pinky
Index (bending)		**25°**		
Index (extension)		-54°		
Middle (bending)	54°		**20°**	
Middle (extension)	**-25°**		-45°	
Ring (bending)		45°		48°
Ring (extension)		**-20°**		-44°
Pinky (bending)			44°	
Pinky (extension)			-48°	

b)

Fig. 2. a) Metacarpophalangeal joint angles for the ring, middle and index fingers. b) Maximum angular distance between the moving finger and its neighbors.

Figure 2-b summarizes the inter-finger angular constraints, giving the maximum angular distance between the moving finger (first column) and its neighbors. The middle and ring fingers have the two smallest values, i.e. the strongest constraints on angular extension. Lifting either of these fingers without also lifting their neighbors requires some effort. We identified eight finger configurations that include such uncomfortable finger combinations (fig.1-b). and removed them from the set of possible chord gestures. The remaining 18 configurations represent comfortable configurations of raised and lowered fingers.

2: *Create additional chord gestures by moving the most independent fingers, i.e. the thumb, index finger and pinky, away from their natural position.*

If we want more than 18 chord gestures, we must modify the relative positions of individual fingers with respect to their neighbors. The hand has natural constraints when determining where fingers can be placed relative to each other. Harger-Ross et al. studied finger independence [2000] by measuring the angular movements of every finger as it was bent or extended. They computed an *individuation index*, which is the ability of each finger to "move when instructed with no accompanying movement of other digits" and a *stationary index* which is the ability of a finger "not to move during instructed movement of other digits". The two indices are highly correlated and allow us to rank fingers in order of their independence: thumb, index finger, pinky, middle, and ring. Harger-Ross et al. found no significant differences between the dominant hand and non-dominant hand. These results let us create additional chord gestures in which more independent fingers, i.e. the thumb, index finger and pinky, are placed a short distance from their normal position. These two rules offer an objective way of generating chord gestures that minimize potential discomfort while maximizing expressivity.

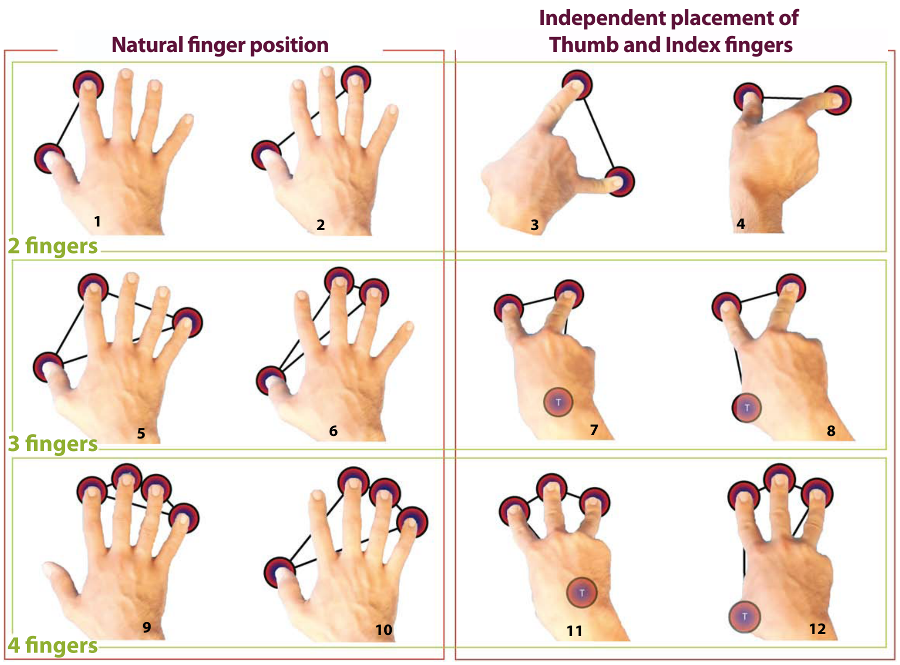


Fig. 3. Multi-finger chord gesture vocabulary

3.1.2 *Gesture Set Example.* Fig. 3 shows a set of 12 gestures based on these guidelines. Matejka [2009] argues that reducing the number of touch points improves comfort. We thus focus on two- three- and four-finger gestures, although one could easily add five-finger gestures. The set includes two gestures for each number of involved fingers and ensures that we do not ask users to raise or lower difficult groups of fingers. Six gestures are based on the "natural" hand position and simply raise or lower different fingers to create two-, three- or four-finger chords. The remaining six gestures take advantage of finger independence and reposition either the thumb or the index finger a short distance from their natural positions. We chose this gesture set because it evenly covers each type of gesture within the design space that results from our guidelines. We use this set to evaluate both efficiency (Exp. 1) and learning of chord gestures (Exp. 2).

3.2 Chord Recognition

Westerman [1999] proposes a recognition algorithm that includes the known position and orientation of the user's hand to determine which finger is in which position. We use a similar approach, but narrow the scale and rotation tolerances.

Rotation and Scale: Rotation independence can be defined either with respect to the table's frame of reference or the user's. Some researchers [Walter 2008] focus on how best to identify the user's position, since, ideally, users should be able to issue chord gestures wherever they stand. We consider rotation only with respect to the user's frame of reference and assume that the user's position with respect to the table is known. This allows us to use rotation as a discriminant feature that both facilitates recognition and expands the number of possible chord gestures. We also account for the relationship of the user's hand with respect to the body when determining rotation tolerance.

We can treat rotation tolerance as a variable when designing a chord gesture vocabulary. If rotation is not a discriminant feature, e.g. when the user's position with respect to the table is unknown, we should set the tolerance to 360°. In such cases, chord gestures will be recognized independently from their orientation. In our gesture set example (fig.3), we set rotation tolerance to approximately 60° (30° clockwise and 30° counterclockwise). This rotation tolerance allows efficient chord gesture recognition and informal observations of users performing gestures from this set suggests that this rotation tolerance remains comfortable to use.

Chord gestures should be also recognized regardless of the size of an adult user's hand. According to [Poston 2000], the hand length of 99% of the population (both men and women) is between 15.9 and 21.9 cm (min/max = 1.37), and the breadth between 7.1 cm and 10 cm (max/min = 1.36). The ratio between the smallest and biggest hands is at most 1.4. This suggests that, rather than full-scale independence, a limited tolerance on gesture size would be sufficient to successfully distinguish gestures that differ in magnitude based solely on hand size. Our sample gesture set uses a tolerance factor for gesture scale of 1.5.

Recognition implementation: Prior to recognizing a chord, we use the Bubble Clusters algorithm [Watanabe et al. 2007] that creates clusters of contact points, based on distance. We set distance thresholds in order to group fingers that belong to the same hand. Hysteresis allows users to explicitly merge two groups of fingers while avoiding most unwanted groupings. We consider each cluster as a potential chord-gesture and compare contact-point clusters to pre-recorded gesture templates. As with the $1 algorithm [Wobbrock et al. 2007], our algorithm is based on point-to-point distances between an input, i.e. a chord, and each template. Unlike the $1 recognizer, our algorithm searches for the template that minimizes the sum of the squared point-to-point distances. We then use this value to compute scale and rotation *a posteriori*. Unlike path-based gestures, there is no order over input points and we use an optimized iteration process to compute distance from the template for all orders of input points.

4. EXPERIMENT 1: GESTURE EFFICIENCY

One strategy for selecting commands on a touch surface is to use a single finger. Most commercial touch-sensitive screen applications offer a tool palette, which requires making a large movement across the surface, followed by a small adjustment to select a specific command. Wu and Balakrishnan [2003] also propose a single-finger context-sensitive menu that displays items in a pie menu around the finger.

We propose an alternative strategy: multi-finger chord gestures to invoke commands. The user places two or more fingers in a particular configuration and places them simultaneously on the touch surface. Unlike tool palettes, chord gestures can be issued in place. Unlike Wu and Balakrishnan's menu, chord gestures can be executed in one step. For our first experiment, we chose to compare chord gestures to tool palettes, since both involve "atomic" gestures with a single touch to the surface instead of multiple steps. Tool palettes are also the most common method for interacting with touch-sensitive surfaces. Experiment 1 focuses on evaluating the efficiency of the *mechanics* of chord gestures, with simulated experts. We measure the cost of moving to a menu versus the mental preparation necessary to perform a gesture in-situ.

4.1 Method

We created two command techniques: a tool palette with 12 items and a set of 12 chord gestures. We used the 12 chord gestures from our example gesture set. These 12 gestures are representative examples that uniformly cover the design space resulting from our guidelines. In order to examine expert performance, we displayed exactly how to execute the command (what and where to touch) at the beginning of each trial (fig. 4). This allowed us to simulate expert performance equally for both the tool palette and the chord gesture conditions. For the tool palette, we displayed the color of the item to be selected on the table, with the tool palette was clearly visible to the participant's left. For the chord gesture, we displayed an image of a hand, with the appropriate fingers highlighted, on the table. This image indicates the precise fingering necessary to perform the chord gesture correctly.

Chord set: We used the 12 gestures illustrated in fig. 3, including equal numbers of two-, three- and four-finger gestures, both "natural" and "constrained" (fig. 4-a).

Tool palette: Commands were represented by a 50x50 pixel square, each of a different color. The tool palette was placed on the left side of the screen, centered along the y axis (fig. 4-b).

Participants: 11 men and one woman participated. Eight were novices with no prior experience using a multitouch surface. Four were intermediate users, with limited experience on a touch-based smart phone. All were right-handed.

Apparatus: We used an FTIR-based multitouch table [Han 2005]. Screen resolution was 1920*1080 pixels for a screen size of 1000.0 x 544.0 mm. The experiment was run on a MacPro computer and software was developed in C++ and OpenGL.

Procedure: We conducted a 2x3 repeated measures within-subject design, with two independent variables: *technique* (chord gesture, tool palette) and *start* position (300, 1100, 1900 pixels) calculated from the left edge of the table. Each technique included 12 *commands*, executed by making a multi-finger chord gesture or selecting a colored square from a one-column tool palette. The three start positions were

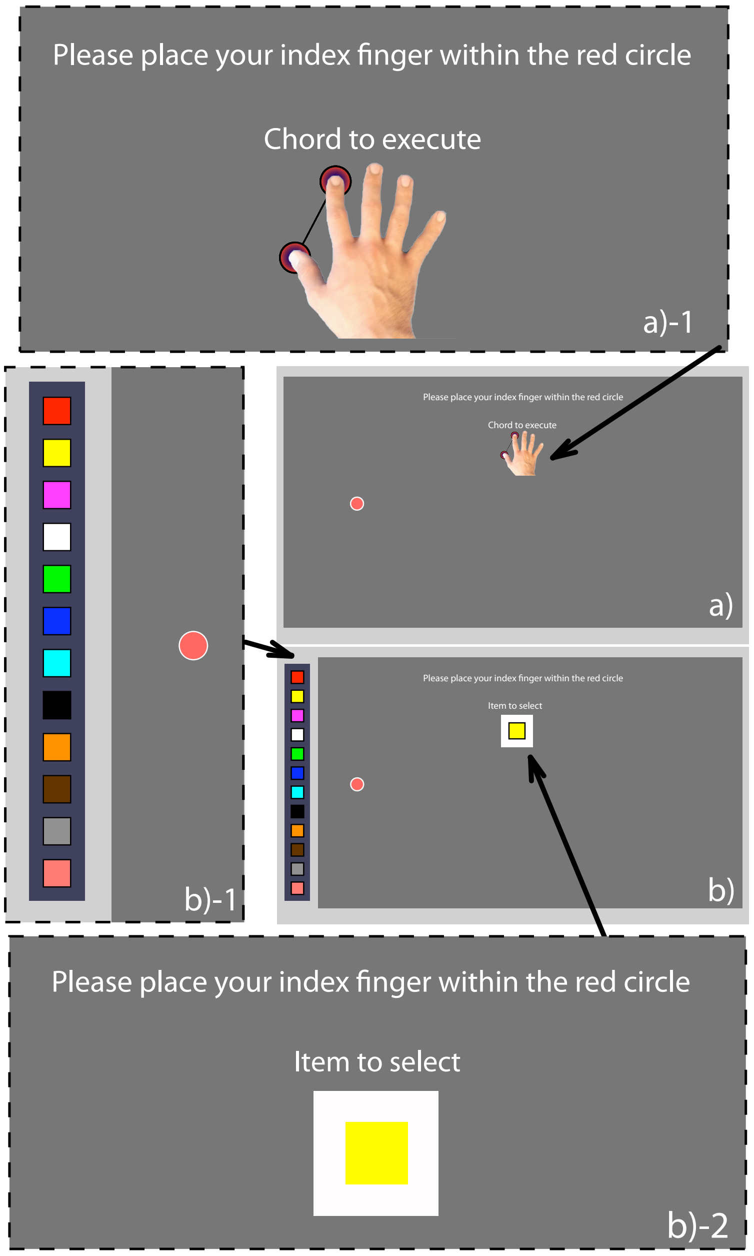

Please place your index finger within the red circle
Chord to execute
a)-1
Please place your index finger within the red circle
Chord to execute
a)
Please place your index finger within the red circle
Item to select
b)
b)-1
Please place your index finger within the red circle
Item to select
b)-2

Fig. 4. a) Multi-finger chord gesture instructions. b) Tool palette instructions. The displayed image indicates which item to select or which chord to execute. The red circle indicates the starting position.

aligned horizontally (Y=540 px) and varied in abscissa. One start position was on the left side of the screen (X=300 px), one in the middle (X=1100 px) and the last on the right side of the screen (X=1900 px). The exact distance from the starting point varied in the tool palette condition, according to the exact position of the item within the palette. Mean distances from the start to items on the tool palette were: near: 325, middle: 1052 and far: 1839 pixels. Participants could reposition their hand from the start position, if desired, to perform chord gestures.

Participants were asked to perform the experiment with their dominant (right) hand, standing in the middle of the long side of the table. Sessions were organized into blocks of 12 items, followed by a pause. Each condition consisted of a 12-item practice block, followed by two 12-item experimental blocks. Participants thus performed a total of 144 trials: 2 techniques x 3 start positions x 12 commands, replicated twice. The order of conditions was counterbalanced within and across subjects. Practice blocks served to familiarize the participants with pointing and making multi-finger gestures on a multi-touch table, since none had ever used one before and only four had ever used a multi-touch device.

Experimental conditions were designed to measure expert behavior, i.e. users know the correct command to execute. For this reason, each experimental trial began by displaying the target action, either the item to select from the tool palette or the chord gesture to perform. The participant begins by placing a finger on the red circle (indicating the starting position), after which a random (1-2 second) countdown begins to prevent the participant from anticipating the start of the trial. Next, the stimulus disappears and the participant performs the required action.

A complete session lasted approximately 30 minutes. Participants were told to perform as quickly as possible. If they made an error, they were asked to restart the trial. We measured task completion time, i.e. the time from leaving the starting point to selecting the menu item or completing the multi-finger chord gesture. We also captured errors.

4.2 Results

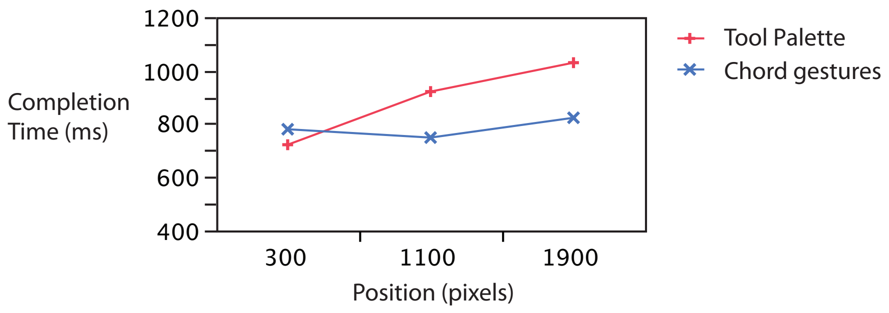


Fig. 5. Completion time for the tool palette (red +) increases with distance while it is essentially constant for chord gestures (blue x).

We performed an ANOVA and accounted for repeated measures by treating participant as a random variable. To calculate completion time, we considered only correct trials, since we were interested in examining expert performance. (Because participants repeated trials in case of an error, we had a complete counter-balanced set of trials.) We observed a significant *Technique* by *Start Position* interaction effect on completion time ($F_{2,22}$ = 53,6, p < 0,0001, fig. 5). When starting the trial from the left side of the screen, i.e. close to the tool palette (mean distance to items = 325px), participants were slightly faster with the tool palette (721 ms) than with the chord gestures (779 ms), although the difference was not significant ($F_{1,11}$ = 1.52, p = 0.24)). As the distance increased, however, completion times were significantly slower for the tool palette. When participants started from the middle of the table (mean distance to tool palette items = 1052 px), participants took a mean of 920 ms to select an item from the tool palette and only 748 ms using the chord gestures ($F_{1,11}$ = 14,45, p = 0.0029). When starting from the right side of the table (mean distance to tool palette items = 1839 px), i.e. furthest from the tool palette, mean completion times were 1029 ms for the tool palette and 822 ms for the chord gestures ($F_{1,11}$ = 11,94, p = 0.0054).

While the effect of position on completion time fits Fitts' law (r^2 = .96), we observed a significant effect of position on completion time for the chord gesture technique ($F_{1,11}$ = 10.22, p = 0.0007). When participants started trials from the middle of the screen, they performed chord gestures more quickly than from either sides (749 ms vs. 822ms and 779ms respectively). This is because participants were asked to stay at a fixed position with respect to the table. If the trial started from the middle of the table, participants could execute the chord gesture quickly and comfortably at or near the start position. When the start position was located to their left or to their right, participants moved their hands to perform the chord closer to the center of the table.

When asked for their preferences, participants were generally positive about the use of chord gestures. Four participants suggested using chord gestures as short cuts for commands they executed often. One participant who had begun his session with chord gestures commented that the subsequent tool palette was much less expressive and powerful. However, all participants mentioned that executing multi-finger chord gestures was new and took more time to get used to. One participant separated the learning process, stating that he would need to learn both multi-finger interaction and multi-finger chords.

4.3 Discussion

Overall, executing a chord gesture was clearly equal or faster than selecting an item from a tool palette, even though it requires a more complex hand configuration and simultaneous finger contacts. Because chord gestures can be executed in place, users do not need to spend additional time traveling to the tool palette. However, an analysis of the error trials revealed that participants made significantly more errors when performing chord gestures, compared to the tool palette ($F_{1,11}$ = 32.93, p = 0.0001).

We were primarily interested in assessing the difficulty of executing chord gestures for expert users and thus omitted error trials from the main analysis. However, we also performed a separate analysis of errors, looking at blocks of 12 trials that used a single technique. Participants had a mean of 3.5 errors per block of chord gestures and 0.1 errors per block with the tool palette. Participants commented that they were already familiar with the tool palette, but needed more time to become comfortable with chord gestures and felt that chord gestures were more difficult to execute than simple one-finger pointing gestures.

Note that this is not due to the users' inability to *remember* which commands were mapped to which gestures, since they had the necessary information at the beginning of each trial. Rather, it is a problem of representation. Users must interpret a static image at a certain distance from the hand, and imagine how to configure the fingers so as to place them on the surface, in the correct position. In addition, users must learn to place all the relevant fingers simultaneously. If the user begins with two fingers of a three-finger command, the system will attempt to recognize a two-finger chord, resulting in an error. Users thus face a basic learning problem.

These results suggest that if we are to successfully incorporate multi-finger chord gestures into the user interface, we will need to provide users with efficient and effective methods for learning and executing them. We know that this is not an insurmountable problem, since experts in other domains are clearly capable of executing multi-finger chord gestures, quickly and efficiently. Our challenge is to create a simple and effective method for teaching chord gestures, to expand the power and range of interactions available on large multi-touch surfaces.

5. CHORD-GESTURE SET REPRESENTATION

We begin by exploring optimal ways for providing users with information about each finger involved in a chord gesture. We need to represent whether a finger is raised or lowered, as well as its position relative to its neighbors.

Design: The most common technique for learning multi-finger gestures are "cheat sheets", e.g. [FingerWorks 2001]. These provide a complete overview of the available commands and their associated gestures and also guide users as to how best to position their fingers to execute each command. Figure 6-a shows how a cheat sheet might represent a multi-finger chord command, using a photograph of a hand, with circles under the relevant fingers. This is also how we represented chords in experiment 1.

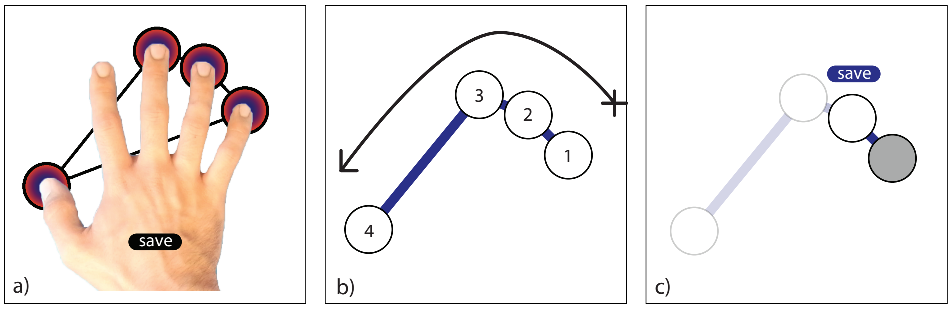


Fig. 6. a) A pictorial representation of a chord. b) The chord is decomposed into an arpeggio, starting with the pinky and ending with the thumb. c) An *Arpege* representation of a chord associated with the *save* command. The user should place his pinky on the gray circle.

A key problem with cheat sheets is the excessive use of screen space. In addition, several participants reported that they had difficulty preparing their hands before placement; the picture of the chord did not provide sufficient information. Novice musicians face a similar problem when trying to decipher chords on a score marked with numeric fingerings. Music teachers find it helpful to encourage novices to first play an arpeggio, laying the fingers down one after the other, before performing the actual chord.

We used this approach to develop *Arpege* which guides novice users, finger by finger, in the execution of a chord-gesture on a multi-touch surface. Figure 6-b shows the optimal order for performing an arpeggio. The first finger is always the one furthest from the thumb, in this case, the pinky. Figure 6-c shows the *Arpege* representation of the chord associated with the same command. It indicates the starting position in gray, with a thick line to indicate the next finger.

Arpege uses a strategy similar to Octopocus [Bau and Mackay 2008], combining feedforward and feedback to provide progressive help to novice users. Feedforward indicates which commands remain available to the user and the finger placements necessary to execute each chord. Feedback indicates which individual finger positions have been recognized thus far, or if the entire chord command has been recognized.

Figure 7 shows a typical *Arpege* sequence. When the user double taps on the table, *Arpege* presents a “fingerprint” with five circles corresponding to the tips of the fingers of the hand. Command labels appear above the circles (*paste*, *copy*, *cut* and *save*) to indicate the appropriate starting finger (fig. 7-a). The user decides to try the *cut* command and places his ring finger on the gray circle (fig. 7-b). *Arpege* then displays the possible alternatives for the second finger of the gesture. Here, the user has a choice between the *cut* and *copy* commands, indicated by colored lines connected to the white dots, each with its own command label. The remaining finger positions are indicated with colored translucent lines that show which subsequent fingers in which locations would result in which commands.

The user continues along the *cut* path, placing his middle finger on indicated dot, which turns gray when he makes contact. At this point, the path for the *copy* chord gesture disappears, since it could no longer be recognized (fig. 7-c). Note that if the user raises the middle finger at this point, *Arpege* returns to the earlier state and the user is free to choose the *copy* command. *Arpege* is not updated if a user misplaces a finger, since the partial input does not match an existing chord gesture. Finally, the user places his finger on the third and final circle (fig. 7-d), which turns gray. The *cut* command label is highlighted to indicate that the chord has been recognized.

Most chord gestures in the literature are triggered at chord down [Matejka et al. 2009]. We used the same approach, so that when the user simultaneously places all relevant fingers in the correct positions the command is executed. If a novice user executes a finger-by-finger Arpege learning sequence, she must then lift all her fingers and explicitly issue a chord down to invoke the command. Expert users simply issue the chord down directly to invoke the command. Of course, Arpege could easily be adapted to handle chord-up, i.e. when all fingers are simultaneously lifted up, to invoke a command. Arpege would then reset when the user lifted all

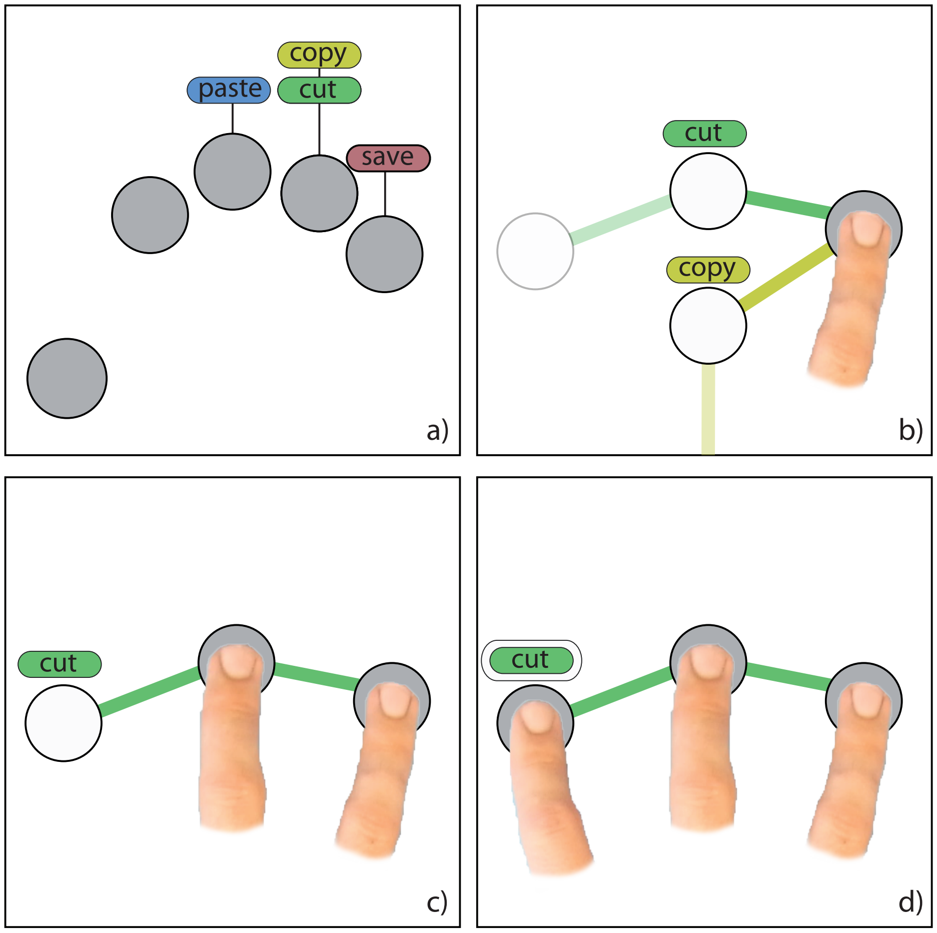


Fig. 7. The user wants to issue a *cut* command, a three-finger chord gesture. a) A double tap produces finger prints, with labels for possible commands. b) He places his ring finger on the *cut* command circle; *paste* disappears. c) He then places his middle finger on the next circle; *copy* disappears. d) He completes the gesture by placing his index finger on the final circle, which executes the *cut* command.

of her fingers.

If the user begins a chord sequence and then lifts one finger back up again, Arpege redisplays the larger set of remaining gestures that were possible before the finger was put down. This allows users to experiment with placing their fingers and see which commands are possible with different subsets of fingers. If the user lifts up all of her fingers without executing a chord, *Arpege* waits

Two chords may share the same fingers in a particular position. For example, in fig. 8-a, both the *open* and *save* commands use the ring and middle fingers. The chords are differentiated by the placement of the third finger, in this case, the index finger is placed in the "natural" position for the *open* command and in a dislocated position for the *save* command.

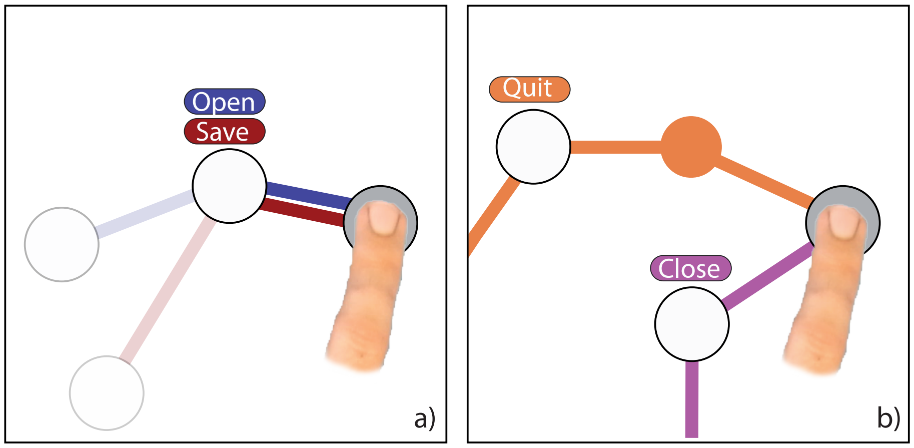


Fig. 8. a) The *open* and *save* commands share the same position for the second finger of the chord. b) The ring finger is already on the table. The orange dot indicates that the next finger for the *quit* command is not the middle finger, but rather the index finger.

Some chord gestures do not use neighboring fingers. For example, in fig. 8-b, the user has placed his ring finger on the table and can see that two options remain: the *quit* and *close* commands. In situations where a finger should not be placed on the table, e.g. the quit command in fig.8-b, the physical distance between the last finger placed on the table and the circle for the next finger clearly indicates which finger to use. User comments suggest that this approach provides sufficient information in most cases. To avoid any ambiguity, the orange circle (Fig.8-b) indicates that the middle finger will remain lifted for the *quit* command and that the index finger should be placed in the white circle, under the *quit* label. The *close* command *does* uses the middle finger. *Arpege* shows that the index finger will be next, in an offset position.

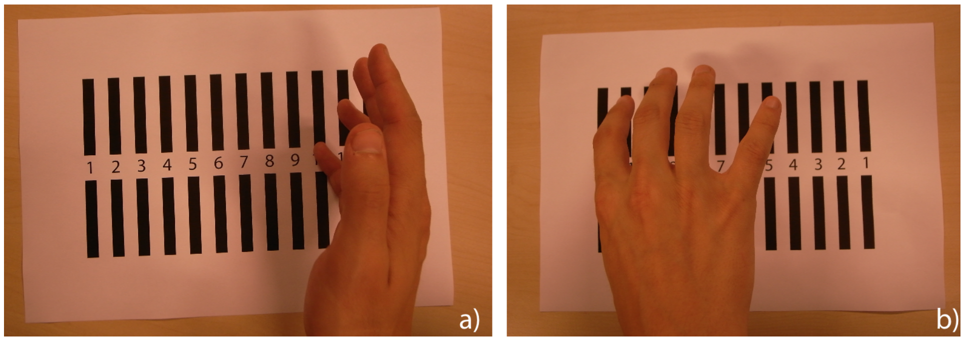


Fig. 9. a) Placing the pinky on the table first reduces occlusion. b) Placing the thumb on the table first increases occlusion. Both thumb and pinky are placed on the vertical segment #10.

Occlusion: One of the difficulties in showing where to place the fingers in a chord gesture is occlusion, in which the user's hand obscures the position of the fingers. *Arpege* addresses this problem by guiding users so that they begin gestures with the finger furthest from the thumb, progressing from the pinky, to the ring, middle and index fingers, and ending with the thumb (fig. 6-b). This results in less occlusion, from the user's perspective, than beginning with the thumb and progressing through the fingers to the pinky (fig. 9). Given the user's position with respect to the edge of the table, we can minimize command label occlusion by placing them above the circles that indicate finger positions.

Implementation of Arpege: We order the points of each *chord template* into a path, beginning with the pinky. Each path consists of a sequence of two, three or four points. We enable partial user input recognition based on a modified version of the algorithm described in [Bau and Mackay 2008]. The key difference is that the number of points in each path is fixed, both for the input and the chord templates, and depends only upon the number of fingers involved. This allows us to easily create scale-independent representations. We estimate the incomplete input's scale relative to a given template by comparing the scale of the input segments to the template segments.

6. EXPERIMENT 2 : LEARNING TECHNIQUES

In the first experiment, participants often made errors even though they had a photograph that showed how to execute the chord gesture. Our goal in the second experiment was to explore ways of helping users learn how to execute multi-finger gestures. We chose the most common training aid, a "cheat sheet", as our control technique, and compare it to the *Arpege* technique. We chose to model our cheat sheet after FingerWorks [FingerWorks 2001], the most commonly used system for representing chord gestures. We show where each fingertip should be placed on a multi-touch surface. Users can easily identify the number and relative positions of fingers in the chord gestures. A picture of the hand specifies which finger goes where (fingering). It also indicates a proper hand configuration for a comfortable chord gesture execution, inspired by our motor rules. In cases where the fingering is unclear, we include the initial letter of the associated finger, e.g. T for Thumb in Fig. 3-11). We are interested primarily in learning rates and performance, since our goal is to help users to quickly move from novice to expert performance.

6.1 Method

Techniques: We used the 12-chord multi-finger gesture set from experiment 1 (see fig. 3-a) and applied the *Arpege* feedforward/feedback technique. Users double-tapped on the table to display *Arpege* at the location of the tap.

We also designed a 875 × 640 pixels "cheat sheet" with a 3x4 array of gesture images and their corresponding labels (fig. 10-b). The cheat sheet appeared at a fixed position (x=145, y=250 pixels), on the left side of the table, to ensure that participants had sufficient space to easily perform multi-finger chord gestures. As with Arpege, this technique was triggered by a double-tap.

Procedure: We conducted a 2x2 within-subjects repeated measures design, with two independent variables: *technique* (cheat sheet, *Arpege*) and sub-vocabulary (two fully counter-balanced sets of six multi-finger chord gestures). Each subvocabulary contained two two-finger chords, two three-finger chords and two four-finger

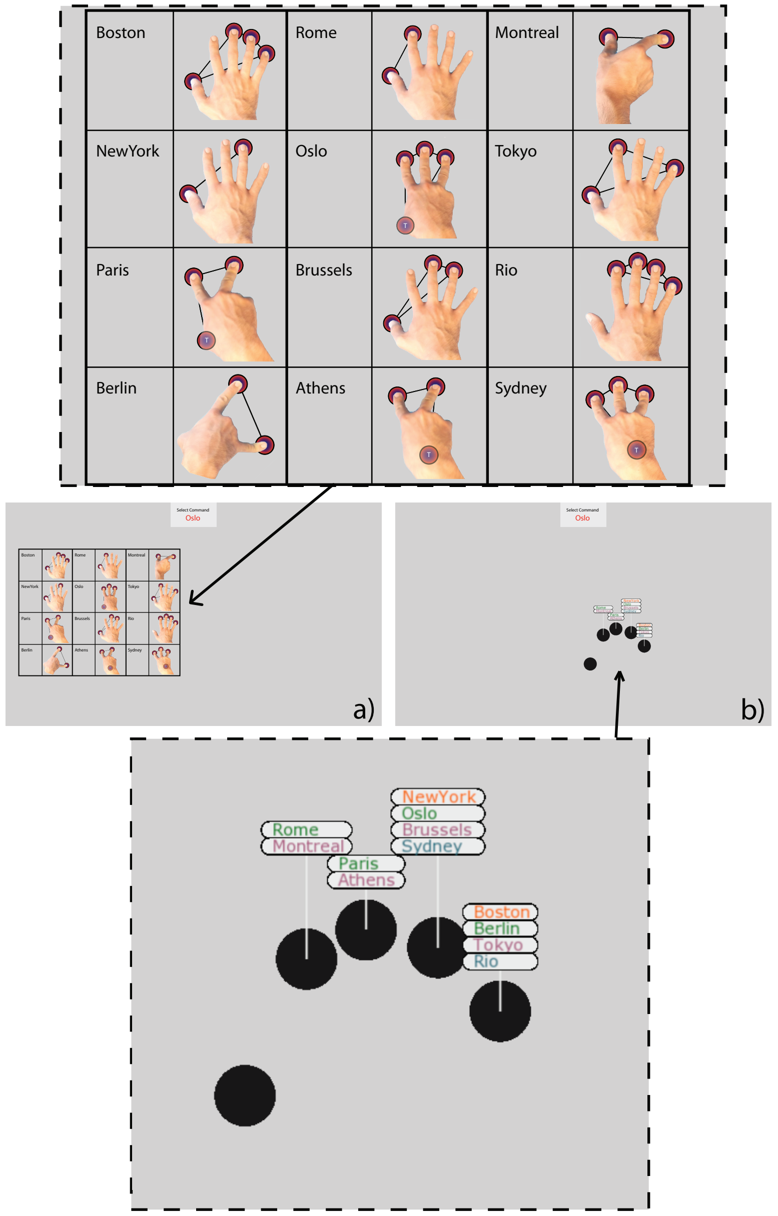


Fig. 10. Each trial displays the name of the command to invoke at the top center of the table. a) The cheat sheet displays all 12 possible chord gestures and their associated commands. b) *Arpege* displays a “fingerprint” with the starting positions of all 12 possible commands.

chords. Half of them were based on the "natural" hand position and the other half on the dislocated hand position. A post-hoc analysis of performance showed that there were no significant differences between the two subvocabularies. Each multi-finger gesture is associated with an arbitrary command, here, the name of a city.

Before the experiment, we trained the participants on each technique (four blocks of six trials). We used a different set of 12 multi-finger chord gestures (also generated from the guidelines) and mapped them to 12 fruit names. This enabled participants to become familiar with both the multi-touch table and executing multi-finger chord gestures in general.

The experiment compared two help techniques, cheat sheets and *Arpege*, using the two subvocabularies. Each subject thus learned six chord gestures using one help technique and the other six using the other technique. Sessions were organized according to technique, with three blocks of 18 trials (six unique gestures, replicated three times). Conditions and blocks of trials were counterbalanced for order within and across participants.

At the beginning of each trial, a circle appears near the bottom-center of the screen. When the participant touches her index finger on the circle, a city name appears and the participant is expected to perform the corresponding chord gesture. If she already knows the appropriate gesture, she simply invokes the chord (*expert mode*). If she needs assistance, she double taps on the surface of the table to get help (*novice mode*), and sees either a cheat sheet or an *Arpege* diagram, according to the current experimental condition. If she makes an error, a message appears and she is expected to try again until she performs the correct multi-touch chord gesture. Note that this incurs a time penalty, but, unlike in a real-world setting, does not also require the user to 'undo' an incorrect command. The next trial begins as soon as the participant performs the correct chord gesture.

We conducted a post-test at the end of each block of 18 trials, with no help available, to determine how well the participants had learned the multi-finger chord gestures of the subvocabulary. We asked participants to perform gestures as quickly and as accurately as possible and to try to really learn the gesture-command associations.

Participants and Apparatus: Three women and nine men participated. We classified eight of them as "novices", with no prior experience with multi-touch interaction and four as "intermediate", with limited experience using a multi-touch surface, such as an Apple iPhone. All participants were right-handed and none had participated in the first experiment. We used the same apparatus as in experiment 1.

6.2 Results

We performed an ANOVA and accounted for repeated measures using the REML procedure in JMP, a statistical package released by the American Statistical Association. When we consider *novice* mode[1], i.e. those for which one of the two help techniques was invoked, participants made significantly more errors with the cheat

[1]Recall that *expert* mode trials are those in which the participant performed the chord gesture without invoking help.

sheet: 39% compared to only 14% with *Arpege* ($F_{1,11}$ = 44.2, p<0.0001) (fig. 11). As one participant observed: *"It is difficult with the cheat sheet to work out the correct hand configuration before laying the fingers on the table to execute a gesture. I needed several tries"*.

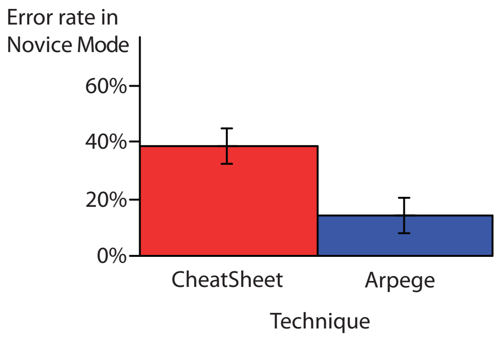


Fig. 11. Error rates in novice mode

For novice mode, the *Completion time* was recorded between the time help was triggered and the end of the trial (fig. 12-green line). Although *Arpege* resulted in almost three times (2.8) fewer errors, we did not observe significant differences in overall *completion time* ($F_{1,10.48}$ = 4.2632, p= 0.0646). Participants spent a mean of 7.4s performing with Arpege against 6.3s with the Cheat Sheet technique. Note that we had explicitly implemented an optimized version of the cheat sheet, using a double tap to display it rather than a round-trip to a menu, as in most applications. This allowed us to compare equivalent trials and ensured that we did not bias the results in our favor.

While these completion times are relatively long, independently of the technique, we found that intermediate users were significantly faster, for both help conditions, than novices with no prior multi-touch experience. If we examine the final block of each technique, when participants were most comfortable with chord gestures and the two help techniques, we find that intermediate users spent a mean of 4.9s with the cheat sheet, compared to 5.1s with *Arpege*. Novice users spent a mean of 6.7s with the cheat sheet, compared to 7.3s with *Arpege*. Although the overall performance of intermediate users was better than that of novices, their relative performance with respect to the two techniques was not significantly different ($F_{1,9.6}$ = 0.06, p=0.8).

The time spent between the first display of the help and the first chord input (whether successful or not) is significantly different between techniques ($F_{1,11}$ = 12.7, p=0.004): Participants took a mean of 4.7s with the cheat sheet and 6.2s with Arpege. This is due to the different strategies adopted by users for each technique. With the cheat sheet, participants quickly tried a series of chords until they found one that worked. Since there was no penalty for error, i.e. invoking an incorrect command and trying to recover from it, this proved a successful strategy. With *Arpege*, participants spent more time placing their fingers to form the chord but were significantly more accurate on their first attempt. This was not only a

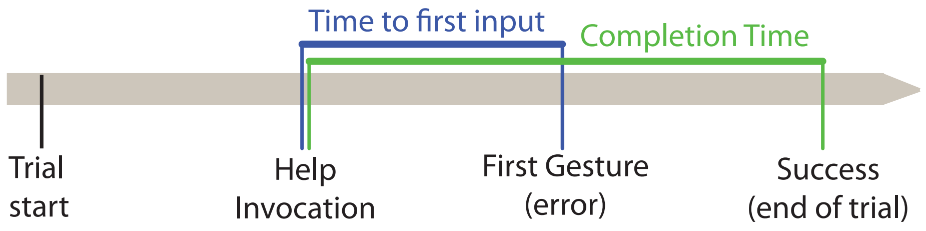


Fig. 12. Two measures: time to first input, overall completion time.

good strategy in the experimental setting, but would also be the preferable strategy in a real-world setting, since errors are costly.

Participants invoked help equally often: 35% of cheat sheet trials and 37% of *Arpege* trials ($F_{1,11} = 0.23$, p=0.63). Since each command is executed three times in a row, most participants used one of the help techniques (*novice mode*) on the first trial and then proceeded to perform the next two trials without further help (*expert mode*).

We found no significant differences in overall learning, as measured by the post-tests: 57% of success with the cheat sheet compared to 54% with *Arpege* ($F_{3,33} = 0.12$, p=0.94). Both techniques successfully aid users in learning to execute chord gestures. Nor was there a significant effect of the number of fingers on learning ($F_{6.66} = 1.84$, p=0.1). Thus, four-finger chords were as easy to learn as two-finger chords.

Although completion times were not significantly different for two- or three-finger chords ($F_{1,11} = 2.02$, $p = 0.18$ and $F_{1,11} = 0.1$, $p = 0.7$ respectively), participants performed four-finger chords significantly faster with the cheat sheet (mean of 6.8s) compared to *Arpege* (mean of 8.2s), ($F_{1,11} = 8.73$, $p = 0.01$). This is probably because *Arpege* becomes slower with each additional finger, whereas the cheat sheet encourages parallel placement of the fingers. In our post-experiment debriefing, participants reported that they found gestures 7 and 11 (see fig. 3) to be the most difficult to perform, even though this had no ultimate effect on learning ($F_{6.66} = 0.5832$, p=0.7424).

6.3 Discussion

In summary, *Arpege* results in almost three times fewer errors than the cheat sheet and encourages users to place their fingers correctly as they form the chord rather than trying a series of incorrect positions until they finally get it right. Although the former strategy may take slightly longer, it significantly increases the probability that the user will execute the desired command, especially important in real-world applications. *Arpege* not only takes less screen space than the cheat sheet, but it can also be invoked in the context of the user's work, rather than forcing the user to navigate to a potentially distant menu.

However, the cheat sheet does have several advantages over *Arpege*. It provides novices with an overview of the full set of available gestures and their global shapes. In contexts in which screen real estate is not at a premium, it probably makes sense to offer both types of help. A novice would then be able to view a cheat sheet for an overview of the possible commands and gestures, at any time, and bring up an *Arpege* representation to guide him finger by finger as he attempts to form the chord.

7. CONCLUSION

This paper addresses the problem of designing and learning chord gestures. It introduced two guidelines for the design of chord gesture sets based on the physiological and motor characteristics of the human hand. The concept of chord gestures was then validated with a first experiment. The results showed that expert users are more efficient when invoking commands with multi-finger chord gestures than by pointing on toolbar items. This experiment also showed that the main challenge for chord gestures is teaching users how to learn and execute commands. We described *Arpege*, a technique that guides novice chord gesture users by providing feedforward on available gestures and feedback on recognition. Arpege guides users finger by finger when learning chord gestures. We showed in a second experiment that users learn the gesture vocabulary as efficiently with Arpege as with a "cheat sheet". Using Arpege is as fast as using a cheat sheet although in our implementation the cheat sheet was invoked with a double-tap, making it more efficient than in real applications. In addition of being as competitive as a "cheat sheet" for learning and efficiency, we showed that using Arpege, novice users made almost three times fewer errors than with the "cheat sheet" technique.

We outline three areas for future work. First, one of our guidelines extends the chord gesture design space by offsetting the most independent fingers away from their "natural" position. We will further investigate this aspect of the design space and study the number of possible gestures that can be generated by varying the relative positions of fingers on a table. We would also like to study specific planar constraints and how they affect hand mechanics on multi-touch tables. We also plan to study the combination of chord gestures with finger movements to control command parameters, similar in principle to Flow Menus [Guimbretière et al. 2005] and Control Menus [Pook et al. 2000] on the desktop. Second, we tested *Arpege* with a set of 12 chord gestures. We plan to study how the techniques scales, i.e. the effect of gesture set size on Arpege's efficiency. Finally, since a user can define the location where Arpege should appear, we also plan to investigate how we can apply Arpege to context-dependent gestures.

OctoPocus [Bau and Mackay 2008] introduced the concept of *dynamic guide*, that improves mouse and pen gesture learning. It provides feedforward on what a user can do, and feedback on how its input has been recognized. We applied this basic principle to chord gestures. It resulted in Arpege, an efficient help technique that makes it possible to include abstract multi-finger chord gestures in the design of multi-touch applications. While this suggests that the concept of feedforward + feedback can be applied to different 2D gesture types, we believe that this basic principle can be extended to gesture that involve more dimensions, adding time for example. Wu et al. [2006] propose gesture phrases for interacting with multi-touch surfaces, i.e. sequences of chord- and path-based multi-finger gestures.

We would like to explore a combination of Arpege and OctoPocus to help users learn and execute this type of gesture phrases. In addition to the time dimension, one could also imagine extending the use of feedforward and feedback to the third dimension and apply it to the guidance of mid-air hand gesture phrases such as those introduced in Charade[Baudel and Beaudouin-Lafon 1993]. We plan to extend the principles of feedforward and feedback to define a design space for the creation of techniques that help users during multi-dimensional gestural input.

REFERENCES

Appert, C. and Zhai, S. 2009. Using strokes as command shortcuts: cognitive benefits and toolkit support. In *Proc. CHI '09*. ACM, 2289–2298.

Bailly, G., Demeure, A., Lecolinet, E., and Nigay, L. 2008. Multitouch menu (mtm). *Proc. IHM '08*.

Bau, O. and Mackay, W. E. 2008. Octopocus: a dynamic guide for learning gesture-based command sets. In *Proc. UIST '08*. ACM, 37–46.

Baudel, T. and Beaudouin-Lafon, M. 1993. Charade: remote control of objects using free-hand gestures. *Commun. ACM 36,* 7, 28–35.

Doo, M., Lyons, K., and Starner, T. 2006. The korean twiddler: one-handed chording text entry for korean mobile phones. In *Proc. CHI '06*. ACM, 718–723.

Elias, J., Westerman, W., and Haggerty, M. 2007. Multi-touch gesture dictionary.

Engelbart, D. C. 1962. Augmenting human intellect: A conceptual framework.

FingerWorks. 2001. igesture products : Quick reference guides.

Freeman, D., Benko, H., Morris, M. R., and Wigdor, D. 2009. Shadowguides: Visualizations for in-situ learning of multi-touch and whole-hand gestures. *Proc. ITS'09*.

Grossman, T., Dragicevic, P., and Balakrishnan, R. 2007. Strategies for accelerating on-line learning of hotkeys. In *Proc. CHI '07*. ACM, 1591–1600.

Guimbretière, F., Martin, A., and Winograd, T. 2005. Benefits of merging command selection and direct manipulation. *ACM TOCHI 12,* 3, 460–476.

Hager-Ross, C., S. M. H. 2000. Quantifying the independence of human finger movements: Comparisons of digits, hands, and movement frequencies. *Journal of Neuroscience*.

Han, J. Y. 2005. Low-cost multi-touch sensing through frustrated total internal reflection. In *Proc. UIST '05*. ACM, 115–118.

Kurtenbach, G. P. 1993. The design and evaluation of marking menus. Ph.D. thesis.

Lee, J. and Kunii, T. L. 1995. Model-based analysis of hand posture. *IEEE Computer Graphics and Applications 15,* 5, 77–86.

Malik, S., Ranjan, A., and Balakrishnan, R. 2006. Interacting with large displays from a distance with vision-tracked multi-finger gestural input. In *SIGGRAPH '06 Sketches*. ACM, 5.

Matejka, J., Grossman, T., Lo, J., and Fitzmaurice, G. 2009. The design and evaluation of multi-finger mouse emulation techniques. In *Proc. CHI '09*. ACM, 1073–1082.

Pook, S., Lecolinet, E., Vaysseix, G., and Barillot, E. 2000. Control menus: excecution and control in a single interactor. In *Proc. CHI '00*. ACM, 263–264.

Poston, A. 2000. Department of defense human factors engineering technical advisory group. static human physical characteristics - hands, in human engineering design data digest. 83.

Rekimoto, J., Ishizawa, T., Schwesig, C., and Oba, H. 2003. Presense: interaction techniques for finger sensing input devices. *Proc. UIST '03*.

Szentgyorgyi, C. and Lank, E. 2007. Five-key text input using rhythmic mappings. *Proc. ICMI '07*.

Vanacken, D., Demeure, A., Luyten, K., and Coninx, K. 2008. Ghosts in the interface: Meta-user interface visualizations as guides for multi-touch interaction. *Proc. TABLETOP '08*, 81–84.

Walter, F. 2008. User detection for a multi-touch table via proximity sensors. *Proc. TABLETOP '08*, 81–84.

Watanabe, N., Washida, M., and Igarashi, T. 2007. Bubble clusters: an interface for manipulating spatial aggregation of graphical objects. In *Proc. UIST '07*. ACM, New York, NY, USA, 173–182.

Westerman, W. 1999. *Hand Tracking, Finger Identification, and Chordic Manipulation on a Multi-touch Surface*.

Westerman, W. 2008. Gesture learning. *freepatentsonline.com*.

Wilson, A. D., Izadi, S., Hilliges, O., Garcia-Mendoza, A., and Kirk, D. 2008. Bringing physics to the surface. In *Proc. UIST '08*. ACM, 67–76.

Wobbrock, J. O., Morris, M. R., and Wilson, A. D. 2009. User-defined gestures for surface computing. In *Proc. CHI '09*. ACM, 1083–1092.

Wobbrock, J. O., Wilson, A. D., and Li, Y. 2007. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *Proc. UIST '07*. ACM, 159–168.

Wu, M. and Balakrishnan, R. 2003. Multi-finger and whole hand gestural interaction techniques for multi-user tabletop displays. In *Proc. UIST '03*. ACM, 193–202.

Wu, M., Shen, C., Ryall, K., Forlines, C., and Balakrishnan, R. 2006. Gesture registration, relaxation, and reuse for multi-point direct-touch surfaces. In *Proc. TABLETOP '06*. IEEE Computer Society, 185–192.