**Alchemy: Dynamic Gesture Hinting for Mobile Devices**

Sus Lundgren and Martin Hjulström
Interaction Design, Department of Applied Information Technology
Chalmers University of Technology | University of Gothenburg,
SE-412 96 Göteborg, SWEDEN
+46 31 772 1042
sus.lundgren@chalmers.se, hjulstro@chalmers.se

**ABSTRACT**

Users of smartphones and other mobile gesture interfaces are often left to explore possible interactions in the interface by themselves; there is currently no generic way to indicate which gestures can be made where. This is problematic. Thus, we propose gesture hinting as a means to deal with this: it serves as a combination of static hinting, dynamic visual hinting and cursor hinting, showing users which gestures are currently available. In short we propose having a set of symbols, one for each possible gesture, which can be combined into gesture hints describing how the user can interact with the part of the interface she or he is currently pointing at. As a proof of concept we have developed *Alchemy*, a gesture hinting application running on iOS, just to verify that it is possible to implement gesture hinting. As a result of this design process we discuss issues related to gesture hinting in general and *Alchemy* in particular; suggest possible solutions; and also point out further issues that need to be taken into account when applying gesture hinting, e.g. temporal effects and gestures for more than two fingers.

**Categories and Subject Descriptors**

**General Terms**
Design, Experimentation, Human Factors.

**Keywords**
Gestures, gesture hinting, Alchemy, smartphones, mobile devices, mobile gesture interfaces, interaction design

1. **INTRODUCTION**

For decades, interacting with a computer has been equivalent with using a keyboard and a mouse, and the current interaction paradigm is based on these two input devices. Accordingly, user interface components (buttons, menus, dialogues) have certain sizes, are arranged in certain ways, and can be interacted with mostly by clicking, sometimes by dragging. However, the advent of smartphones with touch screens as main source of input have changed this, and the upcoming era of tablet computers will even more so. But, despite the fact that there is lots of research on possible gestures, much of it has not yet reached the developers designing and programming applications (or “apps” for short) for these devices. Even more troublesome is that there are no standards or guidelines whatsoever – not even common practice – when it comes to show the users which gestures or interactions they can perform. Currently, it is up to the users to explore the apps in their smartphone or tablet well enough to find otherwise unknown gestures, and as a result, users miss out on possible ways of interacting. There are numerous examples of how gestures unknown to most users are used in an interface – the ones described here can be found on the iPhone. For instance, some apps support a 90-degree orientation change to show data in a new way. This is quite common in apps dealing with weather; if rotating the device, the weather for the upcoming days is presented as curves or diagrams instead of as a table. Another example is that if one has scrolled down on a page or list (e.g. a Wikipedia entry or a Facebook feed) one can return to the top by pressing the screen’s status bar [7]. Similarly, the Swipe gesture (as defined by Apple, [7]) that is made in table cells or lists in order to get a delete-button for the item is not in any way indicated by the GUI. In all cases, these functions are by no means obvious to the user, but are instead something that has to be discovered.

Not only users are confused; so are the developers of apps for smartphone and tablets. The apps on Android Market and iTunes respectively feature a plethora of colorful designs – all solving the same design problems over and over again, but in different ways: how to provide means for interaction, and how to communicate to the user what can be done. Some use standard components to indicate there is more content, such as scroll bars and page components. Others start with an intermediate screen describing interaction. Yet others write interaction instructions in the background. Lastly, some use animations to indicate what is to be done. Arguably, Apple has released their *iOS Human Interface Guidelines* [7], but these do not at all cover this issue. Clearly, there is a need for a common view on how to indicate possible ways to interaction with components on a touch screen. In addition, there is great inconsequence when it comes to applying tilting and/or rotation and/or shaking, i.e. interacting with the devices themselves; only a few apps utilize this, and often in different ways.

Lending from GUI design for desktops we propose the concept of gesture hinting as a solution to this problem; to dynamically and visually indicate which gestures are possible when and where in an interface. Below, we will discuss how affordances and hinting are displayed in desktop GUIs, and how some of this can be (and to some extent already is) transferred to gestural interfaces. We will also provide examples of how many of today’s apps indicate or describe possible interactions, arguing that gesture hinting is a better solution. Lastly we will explore it in our own design *Alchemy*, and discuss related issues. You will find that most of the examples consider the iOS, or iPhone-apps, but the general reasoning and the design solution fit any mobile gestural interface.
2. BACKGROUND

If comparing the normal desktop GUI (Graphical User Interface) with a mobile gestural interface, we find several differences. Firstly, a mouse cursor serves as a precise pointing device that doesn’t obscure the sight for the user. A finger, in comparison, is a relatively clumsy “device” and it’s not recommended that areas smaller than 44 by 44 pixels are used as controls [7]. In addition it also covers parts of the screen. Another difference between the two is that, whereas the cursor is always present on the screen, the finger is only sometimes touching the screen – interacting with the content – and sometimes not. Further, a mouse is used for pointing, several types of clicking, dragging and dropping whereas the corresponding interaction with a touch screen is carried out using gestures. Whereas three decades of GUI-development has resulted in a number of strategies to show what we can interact with on a desktop, the mobile gesture interfaces still lack a generic way of indicating how the GUI surface can be interacted with. There are no standards or guidelines, and only few examples – as we will show – on how to indicate what is unique for gesture-based interfaces: the gestures themselves. This leads to the conclusion that there’s a need for a standardized way of indicating the interaction possibilities to the users. This need has been pointed out by for instance Norman and Nielsen [12], who describe the new gestural interfaces as a step backwards in usability, pointing out the invisible nature of gestures as a reason for this. Clark [2] also mentions this and states that gestures should only be treated as shortcuts and that applications can’t rely solely on gestures since one cannot rely on users finding them. Similarly, Spencer as quoted in [6], says that interpreting gestures isn’t the problem, communicating them to the users is.

2.1 Hinting in GUIs

In 1989, Donald Norman introduced the notion of affordance to the HCI-community, defining it as “the perceived and actual properties of the thing, primarily those fundamental properties that determine just how the thing could possibly be used.” [11, p. 9]. Norman discussed physical artifacts like doors, light switches and video recorders, but nevertheless, this need for similar affordances was acknowledged in the GUI-design community. For example, the first version of Microsoft Word, released in 1989, featured buttons and icons with a distinct 3D sculpturing effect. A few years later, Cooper described the GUI-version of affordances as pliancy; referring to any object or screen area that can be manipulated as being pliant [3]. As described by Cooper et al, pliancy can be indicated in four different ways [4]:

**Static object hinting**: an object always communicates that it can be tampered with, e.g. the three-dimensional sculpting of a button.

**Dynamic visual hinting**: an object changes appearance when a cursor passes over it, e.g. how icons on a toolbar are highlighted when the cursor passes.

**Pliant response hinting**: an object changes appearance when interacted with; e.g. visually indicating that a button has been pressed.

**Cursor hinting**: the cursor changes appearance depending on what it passes over; e.g. when passing over the edge of a resizable widow it changes into two arrows to indicate this.

The simplest type of hinting – static object hinting – is easy to create with more or less discrete 3D sculpting. Over the years, this type of hinting has become less explicit; e.g. icons have evolved from the clumsy button-like icons in older versions of e.g. Word to small slick squares with a discrete gradient shadow on the background. However, static visual hinting like this can be allowed to be discrete because it can be supported with dynamic object hinting and/or pliant response hinting, which is almost always the case. In combination these three types of hinting work well to indicate that, and how, a control can be interacted with. Whereas these types of hinting are standard in all desktop GUIs, they are often lacking in applications on mobile devices. Arguably, most of the standard controls utilize static object hinting (e.g. by using gradient backgrounds to indicate a button or arrows on menu items with sub-menus) and pliant response hinting (buttons and menu choices change color when selected for instance), just as in any desktop GUI. However dynamic visual hinting and cursor hinting are lacking. As for dynamic visual hinting, the finger obscures it in most cases, and additionally, there is hardly any difference between touching an element aiming to click or tap it and touching an element aiming to do a “mouse-over”. As for cursor hinting, there is no visual cursor on screen – the finger is the cursor. Thus, the needed enforcement of static object hinting is lacking in gestural interfaces.

2.2 Current Research on Gestural Interfaces

Naturally, the interaction design and developer communities are brimming with new possible ways to interact with touch screens, especially when it comes to suggesting gestures. E.g. Ronda et al [18] describe how the vocabulary of gestures can be extended by distinguishing between different types of thumb movement. Roth and Turner [17] instead utilize the inactive bezel (the outermost frame) on the smartphone screen as a static button. As for the “mouse-over”. For as cursor hinting, there is no visual cursor on screen – the finger is the cursor. Thus, the needed enforcement of static object hinting is lacking in gestural interfaces.

**2.3 Indicating Gestures in Apps Today**

We have investigated strategies for explaining to the users how to interact using gestures other than using static visual hinting to
indicate that something is a button and thus can be pressed. Below we describe three common design solutions that can be found in many apps today; some 200 iPhone apps were harvested for solutions of which all fall into the following three categories; to indicate more content using standard components; to give interaction instructions in writing; or to use animations to show to the user what he or she should do.

Indicate there is more content. Two standard iOS components indicate that users can make a scrolling gesture either vertically or horizontally. Examples of the vertical scroll bar can be found in Wikipedia, Google, Gmail Facebook and most other applications presenting lists, text or posts in a column. The horizontal scroll bar is less common but can be found in Safari, MindMeister and other apps with large continuous areas or canvases. Some applications have several screens organized horizontally that the user can navigate through discretely (as opposed to the continuous scroll made with the scrollbars). The page control is often used on those situations. Not only does it indicate that there are more pages but it also tells the user the current pages position. Examples include the home screen of iOS devices, iBooks and DinoMixer. These two controls, together with the unlock-control for unlocking the phone (this is further described later and can be seen in Figure 4), are the only examples of attempts to indicate gestures in the standard components in the iOS.

Give written instructions. The least “hinting” way of indicating how to interact with an app is to provide written instructions. This is often done in a separate view accessed from the menu or from an info or help button, as in Angry Birds, Blowfish, Bloons, Smack Talk, Auditorium and many other games that utilize gestures.

Figure 1: To the top, DinoMixer utilizes both a scrollbar to navigate vertically, and a page control to indicate that there are more pages.

Another solution is to use alert views – i.e. pop-ups – to communicate with the user, an approach used in Photoshop Express where it is used to explain how to work with some tools.

Yet another approach is to explain possible interactions in the background of the GUI, as in for instance Swanko Labs, Cut the Rope, and MIG. If there are many ways to interact, tutorials are often used, as for instance in the games Grimm and Parcel Panic. An intermediate version is to halt the proceedings before something new (e.g. a new level) is to appear and provide

Figure 2: Help screens from Angry Birds (left) and Doodle Truck (right).

A quite different approach is to show the user the gesture as an animation. One example is the game Fruit Ninja where the user is supposed to slice fruits by sliding their finger over them. The entire interfaces features this type of interaction, e.g. menu items are not tapped as in the normal case, but instead “sliced” with a sliding motion. If the user does not understand this, but tries to tap instead, a hand that animates the correct slicing gesture appears as an aid to the users. In this way, it very promptly indicates how to interact with the rest of the GUI. A similar approach is taken in Block Shooter.

In Auditorium, the objective is to create a soundscape directing waves with arrows. The result is often beautiful to listen to, and as a result, the game does not automatically continue to the next level. Instead a discrete hand appears, pointing to the part of the screen where the user may push once he or she is done listening. This animation fits very well with Auditoriums otherwise very sparse look and feel; a button would have been intrusive. In this, Auditoriums hint is very discrete and unobtrusive.

Figure 3: Instructions in the background; Cut the Rope (left) and Swanko Labs (right)

instructions. Examples include games like geoDefense, Doodle Truck, Zombie Cafe and Physics.

Use animations. One of the most flagrant examples of animated hints is the control to unlock the iPhone: a button with an arrow on it, and a text saying “slide to unlock”, augmented with a scrolling highlighted moving from left to right. A similar take is used in Flipboard, an app that acts almost as a start page on the Internet, connecting several services such as Facebook and news in a common app. When it is opened an arbitrary image is displayed. To get to the main menu the user is supposed to swipe – or flip as it’s called in the app – from left to right on the right side of the screen. To indicate this, Flipboard has a text that says: “flip” with the same moving highlight as the “slide to unlock”.

Similarly, the game Monster Dash uses fading controls. In the game, the user can either jump (by tapping the left side of the screen) or shoot (by tapping the right side). This is thoroughly hinted in the beginning of the game but as the game progresses the hinting will fade into the background until its completely gone. If the user misses the controls too often or is hesitant, they reappear. Another example is Spawn Sparkle, where users toy with cascading lines by guiding them with their finger. If a second
finger is placed on the screen, the control for changing attributes of the lines appears and indicates a pinching motion.

2.4 Issues Related to Current Strategies
For the currently used ways to indicate or inform about gestures, some are problematic. For instance, providing written instructions in the form of a pop-up interrupts work. If the hinting instead is in the form of instructions written in the background, it is always present. The user can see it without having to interact with the screen, which is an advantage, but the downside is the space issue; one of the biggest struggles in design for the small screen of smartphones is its limited screen size [7], and in many situations there’s simply no room for static hinting of this type. If using static hinting too extensively there is also a risk that the screen ends up being cluttered, drawing the attention away from the actual task. Also, using written instructions may be accurate and hard to misinterpret, thus pedagogic, but after a few hours of use, the solution is neither fast nor elegant; Cooper discusses a similar situation when describing how users use menus (text) to browse though and learn an interface, whereas experienced users have learnt to utilize the icons, which opens up for a faster workflow [4].

Dynamic animations have the advantage of only being visible when needed, but here, the big issue is when to display the hints, or how they should look to best communicate the gesture. There are no guidelines or suggestions regarding how to do this. Arguably, there are a few examples of built-in iOS controls that hint which gestures to use, e.g. the scroll indicator. The benefit of having the hinting built into the iOS is of course that it will look and behave the same in all applications, meaning that users will recognize it, instantly knowing how to interact. This type of consistency is seen as one of the main virtues in traditional GUIs [4, 10, 15].

As for the solutions suggested by researchers, Wigdor et al’s [19] design was designed for large surfaces. As for Sanchez [19] suggestion to use sound, it is very promising albeit the use of sound can be problematic in some contexts, like in public environments, partly because the user may not hear the properly, or because they might disturb others. Yatani et al [21] have combined gestures and information visualization, and have thus specialized on the limited problem of selecting small items on a crowded screen. This too, is a promising and interesting research which however does not address the issue investigated here; how to provide information on what gestures can be used in general.

3. GESTURE HINTING
The conclusion of the above is threefold: that mobile gesture interfaces lack ways of communicating how to interact with them; that textual instructions are a clumsy solution to this; and that there is a lack of guidelines for dynamically indicating gestures. We wish to introduce gesture hinting as a possible solution to this. As the name suggests, it is similar to static object hinting, dynamic visual hinting, pliant response hinting and cursor hinting used in regular desktop GUIs. In this, it will replace – and be a combination of – dynamic visual hinting and cursor hinting, and thus serve to support the static object hinting and pliant response hinting already used in today’s smart phone GUIs.

We define gesture hinting as symbols that indicate which gesture(s) can be made in a gesture interface. These hints can be either dynamic or placid as follows: Dynamic gesture hinting consists of dynamically appearing and changing symbols that follow the finger, indicating which gesture can be made on the surface currently touched. Placid gesture hints consist of symbols which are independent of the moving finger(s). They may be static in that they are always present, or semi-static in that they may fade in and out of view, however always in the same location. In this paper, we have only explored dynamic gesture hinting, since that presents the most interesting design challenges.

A gesture hint only indicates what can be done with a control or in a specific area, not what the effect of the action is. As a result of this limitation, it ought to be possible to create a language of generic gesture hints. Agreeing with Cooper [4], we believe that this language ought to be idiomatic – i.e. not necessarily intuitive, but easy to learn, and once learnt never forgotten. Most interface controls are already idiomatic since they do not have any equivalents in the real world; scroll bars, menus, hyperlinks, drop-downs etc. [4]. The advantage of idioms is that they can be made so generic that it will fit most applications, thus serving as a well-needed standard component. In the next section we describe our suggestion on how such a system might look and work.

4. ALCHEMY: A DESIGN EXPLORATION
In order to explore the concept of dynamic gesture hinting, we designed Alchemy. The aim was not to find a perfect solution at this stage, but rather to create a proof of concept, and to find, explore and discuss issues related to gesture hinting. The name comes from the fact that we combine different symbols into others, thus giving them a richer meaning.

In short, our solution is to overlay semi-transparent gesture hints on top of the interface – any interface – which means that the design (graphics and code) can be used together with any application. Since some surfaces allow for more than one gesture (e.g. tapping and swiping, as when going through the list of one’s mails in Gmail), we have created a set of suggested icons, one for each gesture, which can be combined, resulting in a number of composite gesture hints covering any possible combination of gestures.

Figure 5: The symbols for Tap, Double tap and Swipe are combined into a composite gesture hint.

In short there were three major issues to take into account: which gestures to visualize, how, and lastly how to implement this. Below, all three parts are discussed, and a final example of how it could work is described.

4.1 Gestures: which ones and why
Apple lists eight finger gestures [7]. From a code-point of view however, only seven different main gestures can be discerned [5]. Out of those seven Android can detect five [1], the two missing being the two finger gestures pinch and rotate. We have, however, focused on the seven defined by Apple, firstly because our proof of concept was developed for iOS and lastly because the two finger gestures introduces interesting problems, which is discussed later. The seven gestures are as follows:

- **Tap**: A tap can either occur when touching the surface or when releasing it, but from a visualization point-of-view these interactions are so similar that they can be seen as one and the same. Tapping can be made with one or two fingers.
- **Double tap** and **multi tap**: Strictly speaking, these are just multiple instances of tap. For the user however, they are different things. Tapping can be made with one or two fingers.
**Swipe:** This is a horizontal and/or vertical movement. The iOS can make a difference between all four directions, e.g. only reacting on a left-to-right movement as opposed to right-to-left. This corresponds to Android’s drag gesture.

**Drag:** This is a movement in any direction, typically used when drawing, panning or dragging. It corresponds to Android’s scroll gesture.

**Pinch:** This is a two-finger gesture that is used for zooming in or out: moving finger tips closer together zooms out, moving them further apart zooms in.

**Rotate:** This is a two-finger gesture, a turning motion used to rotate a view.

**Touch and hold (long press):** When the user touches the screen and holds it for a longer time.

### 4.2 Possible Design of Graphics

Starting out with the premise that gesture hints are combinations of different components, we have explored three tracks, all utilizing the idea of a center circle as basis, with additional arrows and other icons to symbolize different gestures.

In all designs below we have chosen to use a circle as the base for all symbols. The main reason for this was that the circle would somehow indicate the area that the gesture hint applies to. But – should this circle also symbolize tap, since the tap is inherent in any other gesture? Some may argue that some GUI-controls, e.g. sliders, only react to sliding, but not tapping. Should one then design an extra component to indicate just that the finger is touching the screen, but not tapping it? We however believe that conceptually, the difference between a tap and to-touch-to-move is very small, and that this justifies the (beneficial) choice to have one symbol less.

When it comes to the placement and size of the gesture hints, we have explored several approaches.

![Figure 6: The three main design ideas (natural size), all indicating possible idioms for Drag (the wavy line out to the right that all three possess) plus Touch and hold (the rest of the symbol).](image)

In one design (the leftmost in Figure 6), we imagined a small hint showing above the pointing finger, indicating the possible interactions. The downside was that the user might misinterpret it as indicating the possible interactions for the area where the symbol was shown, as opposed to where the user was actually pointing. Another variant (the middle in Figure 6), was to depict the possible interactions “from the side”, but it suffered from the same problem, and in addition, some of the components might obscure each other. Both of these designs have the benefit to be quite small, however not that small; in order for all six icons to be recognizable (especially on cluttered backgrounds), they still needed to be comparably big. The third idea (the rightmost in Figure 6) was instead to surround the pointing finger with the circle. Arguably, such a symbol is quite large, but it has the benefit of being very intuitive, undoubtedly indicating what can be done with the pointing finger. Thus, this approach was explored further.

This brought us to the issue of actually designing the different symbols. A series of suggestions were tested; both in the form of the worst-case gesture hints, (i.e. combinations of all symbols) to ensure that all symbols would fit together; and in the form of more likely composites. To ensure that they would be visible in any environment they were tested on a variety of different backgrounds. As a result, the icons are white but slightly transparent, having a drop shadow, which makes them visible even on cluttered backgrounds (see Figures 8 and 9). What could be learnt from these design tests was that details like line width, opacity, size of icons etc, really matter, and this is by no means investigated fully. Anyhow, our final design was a combination of circles, lines and triangular arrows.

![Figure 7: The current set of gesture hints](image)
But – as soon as two-finger gestures are to be indicated, a series of issues emerge. Firstly, this second finger needs to be indicated somehow. Arguably, one could refrain from indicating it, which would break coherency, or possibly just add yet another attribute to the existing hint, but this might not be obvious enough. Adding a second hint (i.e. a full circle with components) takes a lot of space into account, increasing the odds that it won’t fit within the actual area that allows for two-finger interaction. It also brings the risk that the user is led to believe that he or she has to place the second finger exactly there. Our solution was to create a smaller, intermediate hint, indicating that a second finger could be placed; once placed (and wherever that is) – the normal hint would appear around that fingertip too. See Figure 9 for an illustration of this.

But, if a second indication is to be shown, one must decide where. If always shown in a static place, say slightly below and to the left (to accommodate right-handed users), it might end up in an area that does not support this kind of interaction. Certain flexibility, regarding where this second-finger hint should appear, is needed. Currently our suggestion is to have four slightly different versions of the design, one for each possible placement, however we are aware that initially this might confuse the users.

Despite a quite thorough rationale, there are still a lot of issues related to the design. One is, of course that despite our best efforts, it has to be quite large, given the size of an average male finger, something that is countered by making it opaque. But as shown in fig 6, earlier, even an icon not encircling the finger needs to be comparably big.

Another issue is related to the arrows per se, e.g. the four arrows indicating swipe-like motions either horizontally or vertically, simply because that is the direction one is supposed to move one’s finger in. However the arrows for pinch are at 60 and 240 degrees, but actually a pinch can be made in pretty much any direction – will the symbol thus lead the users to believe that a pinch has to be made in this exact angle? We hope not so.

Additionally the general problem with the user’s finger obscuring part of the screen is still valid. Again, for using one-finger gestures the design works fine, since all attributes except the 270-degree arrow are in the upper half. But, for two-finger gestures, what should be the default placement for the intermediate second-finger hint? For a right-handed person, using the index-finger to touch the screen, placing the hint to the lower left would be natural, because the user would then place his or her thumb somewhere there and could start pinching, rotating etc. But, this does not work as well for the left-handed user, pointing with the index finger, which then obscures the intermediate symbol.

4.3 Code

There are at least three different architectures to choose from when realizing gesture hinting. The ideal would be to include it as a part of the frameworks, since it would then turn into some kind of de facto standard. Another approach would be to develop an application that can be installed on the devices per se and run in the background and add an overlay to the application currently running. The last solution is to develop it as a framework that developers could use in their own project, running in the background picking up gestures and add an overlay with the gesture hints.

To verify that our ideas were possible to implement, we wrote an Alchemy-application for the iOS, since we then could utilize its built-in gesture recognizers, and thus easily detect and present more gestures than what would have been possible on Android. Our code took the form of an independent framework, this since the other approaches are not supported by iOS. Of course this means that it is up to developers to actively use it, which decreases the chances of it becoming de facto standard.

An overall issue with not only this design, but any system that tries to recognize general gestures, are custom made-gestures; something already supported in Android [9]. Of course Alchemy has no code or visualizations for them. The current code can only recognize the standard components and the gestures they support. This means that if a developer wants to create custom gestures, and still use Alchemy, she or he needs to add them to the Alchemy code base, also adding the related new graphic components for it, and these need to fit the other components in such a way that any combination is possible. This is of course unfortunate, but something that all standards suffer from.

5. DISCUSSION

Apart from the issues related to graphic design and placement, there are some overarching issues related to our suggested design. Most importantly, the temporal aspects of the hinting need to be considered. It is probably not desirable that the hints are visible at all times; they should appear when a user learns an interface, or if the users seem to have forgotten a certain gesture, as in Monster Dash, where controls fade back into view if the user keeps missing them. Thus, finding out when to hint is a delicate issue. There are several possible approaches; to indicate all possible gestures the first minutes of use, or to hint as soon as the user touches a new area, and removing the hint once the user has made the gesture. Or, dynamic gesture hints could always be visible but only for a very short time, tuning in when a new type of area is entered and then and tuning out again. Or, they could appear if the user seems to hesitate. There is no general design decision to be made here; which way to go depends largely on the type of app. Apps that are not used very often, perhaps once or twice a year probably benefit from more hinting, whereas extensive hinting would probably be perceived as being extremely annoying in an everyday app. This issue is a project in itself, aiming to find out with parameters or settings that need to be provided to developers, and how these can be adapted to different types of apps and contexts of use. Possibly dynamic gesture hints may take the same path as static visual hinting in GUIs, turning more and more discrete until they are so non-intrusive that there is no need for fading in or out.

Figure 8: Alchemy running on an actual device.
a) The user touches the Safari icon. *Alchemy* indicates that there are three possible gestures: Tap, Swipe (horizontally only) and Touch and hold. (Note that this requires that *Alchemy* becomes part of the iOS.)

b) After some intermediate steps, the user has reached a web page. She touches the toolbar, and *Alchemy* indicates that Tap is the only available gesture. (Note that this requires that the Safari app incorporates *Alchemy*.)

c) The user now touches the web page, and *Alchemy* indicates that tapping, double tapping, swiping vertically and pinching can be done. The intermediate hint for a second finger has to be placed top right.

d) The user is now touching the web page with two fingers, and thus there are two full circles on screen.

e) The user has pinched, zooming in on the page. Note how the vertical Swipe hint has changed to a Drag hint, indicating that she can scroll/drag in any direction since the zoomed in mode allows for this.

f) The user has now released the second finger and scrolled to the left. The intermediate hint is now placed bottom left, since this is default.

Figure 9: A possible use scenario, and how dynamic gesture hinting could be shown if incorporating *Alchemy*. 
A valid critique is that Alchemy has not been user tested yet. However, as stated, we aimed for an idiomatic interface rather than an intuitive one. From that point of view there is no need to test how easy it is to understand the hints as such. Moreover, this project aimed more towards suggesting the use of gesture hinting and explore and discuss the issues related to this in relation to our design proposal. Thus, user studies were not a part of the scope for this initial part of the Alchemy-project, which of course only means that it is future work awaiting us, together with exploring how users should best learn the hints; as with any idiomatic component, these need to be taught rather than intuited.

Also, gestures for more fingers are a most likely upcoming feature, especially on tablet computers like the iPad and the Galaxy Tab. In that case, the collection of gesture hint icons might have to be tweaked accordingly, but the general concept still stands. It should be able to tackle multi-finger gestures in the same manner as two-finger gestures; it’s of course hard to tell without knowing what they might be.

One could question if visual hinting is the best or only way to go. Others senses could be used, for instance sound. However sounds are sometimes used by apps anyhow, and it would be impossible to foresee how the hint sounds would work together with any custom sound. Moreover mapping sounds to certain gestures is harder than mapping visual symbols; after all our visual memory is much more developed than the auditory memory [14]. Additionally, sounds are transient, and if the user misses the hint due to other sounds it is “gone”. Another alternative is to use vibration. However some apps use vibration to give feedback, and in addition, mapping vibrations to gestures is probably even harder than mapping sounds.

6. CONCLUSION

As a response to the need for improved use and understanding of mobile gesture interfaces, we have created Alchemy, a system that dynamically detects and visualizes which gestures, or combination of gestures, that are available when touching a certain area in an iOS app. In this, Alchemy exemplifies what we call dynamic gesture hinting – to follow the finger and on the fly indicate which gesture(s) can be made on the surface currently touched. Moreover it highlights a series of issues related to the design of gesture hints, thus opening up for further attempts to solve the issues with poor hinting in mobile gesture interfaces, hopefully making way for a generic way of visualizing gestures.

7. ACKNOWLEDGMENTS

We would like to thank Staffan Björk, Erik Fagerholt, Dimitrios Gkouskos and Olof Torgersson for giving input on the design as well as providing some examples of good and bad examples.

8. REFERENCES

6. Ginsburg, S. Designing the iPhone user experience: A user-centered approach to sketching and prototyping iphone apps. Addison Wesley, 2010