Overcoming Occlusion-Problems on Touch Screens

Julian Böhme

Abstract— In this paper I want to elaborate on common problems and disadvantages on touch-based interfaces like smartphones and tablets. I will talk about occlusion-problems caused by the hand, arm, and the finger of the user and about touch imprecision caused by finger occlusion. Further I will describe different methods that have been researched to handle occlusion caused by the hand and arm of the user, as well as methods that deal with avoiding finger occlusion and the resulting touch imprecision.

Index Terms—occlusion-problem, mobile devices, touch-screen, occlusion-aware interfaces, touch imprecision, precise target acquisition

1 Introduction

In recent years interaction with touch screens has for most people become part of their everyday life. 378.000 iPhones are sold each day. As a comparison: Each day 371.000 babies are born [3].
From the year 2012 to 2013 tablet sales increased by 54.1%, from 2013 to 2014 they increased by 29.1% [2]. The number of tablet users is expected to surpass 1 billion in 2015, which are more than twice as many users as in 2012 [2].
But as popular as touch screen devices might be, they bring some disadvantages and problems, non-touch based devices don't have to deal with. When using touch screen, we are confronted with the problem of occlusion of the screen by the user during user interface interaction. As soon as the user wants to interact with the device, he needs to position his hand or arm on or above the screen, occluding some areas that might contain important information. For example: The user wants to tweak a parameter using a slider, while observing the parameter or the result on the screen. If the slider is not positioned on the bottom of the screen, the user might occlude the area that changes (the area he wants to watch) while using the slider *(fig. 1)*. This is known as the occlusion problem.

In this paper, I want to differentiate between two types of occlusion: Hand- / Arm-Occlusion and Finger-Occlusion.
The former simply occludes an area on the screen, the latter does not only occlude the area but also, caused by the occlusion, creates touch imprecision: The user tries to hit a target and due to the occlusion, it's not only hard to hit the target, it can also be hard to even know whether or not the target has been hit, because necessary feedback the button would give, if pressed, might not be seen due to the occlusion.
So Finger-Occlusion creates more than just one problem: There is the occlusion itself, and problems caused by the occlusion, which are touch imprecision and possibly insufficient or no feedback at all. Smartphones have a way of handling the feedback problem: For example, if the user types a message the pressed buttons are shown above the finger as feedback to the user that the button is pressed and which one is pressed. Different techniques have been studied and developed to resolve problems like occlusion and high-precision touch input.

2 The Occlusion-Problem

To solve the occlusion-problem we try to design occlusion-aware interfaces, which are defined by Vogel [4] as follows: *"We define occlusion-aware interfaces as interaction techniques which know what area of the display is currently occluded, and use this knowledge to counteract potential problems and/or utilize the hidden area."* [4] In a study Vogel et al. found the following problems to be caused by occlusion [4]: Inefficient movements, missed status messages, missed previews, and occlusion contortion.
Vogel et al. [4] developed a scalable geometric model to represent the area occluded by hand and arm. The user can adjust the shape of the model to match his hand and arm. Vogel et al. [4] identify changes on the screen, to find out which information on the screen is to be considered important, and which of the occluded information should be displayed in a callout.

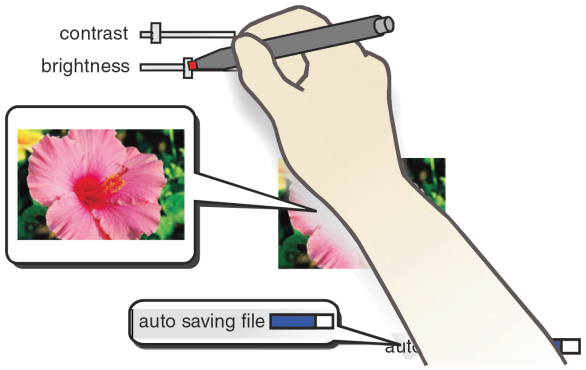

Figure 1. Example of an occlusion-aware interface

3 Occlusion on Multi-Touch Tabletops

In another paper, Vogel et al. examined the shape of hand and arm occlusion on multi-touch tabletops. They created occlusion templates for designers to use and to consider when designing interfaces to avoid occlusion. They also wanted to show that it is possible to use one model for a wide range of postures. The resulting model can be used as a configurable real-time occlusion model for future work [6].
Vogel et al. conducted an experiment, in which participants were to perform different multi-touch gestures, while their hands were recorded by a head-mounted video camera. Then key frames were extracted and occlusion shapes were isolated. The experiment examined 3 main types of interaction movements: Tapping, dragging and object transformation. The participants had 3 tasks to fulfill:

- Tap Task: Participant has to touch a circular target for 333ms
- Drag Task: Participant has to drag a circular target from the center of the screen to one of 8 circular docking locations, using either 2 or 5 fingers.
- Transform Task: Participant rotates and scales a circular target. The target has to be rotate in such a way that a pin aligns with a "key" and the target has to be scaled until it fits into a specified circle.

- *Julian Böhme is studying Media Informatics at the University of Munich, Germany, E-mail: j.boehme@campus.lmu.de*
- *This research paper was written for the Media Informatics Proseminar 2015.*

Some of the main findings of this experiment, considering occlusion shapes, were:
Tapping and dragging have similar shapes, but transformation shapes are different. Different users might use different postures but posture shapes within users are similar. No distinct differences between left- and right-handed people, and dominant and non-dominant hands could be found [6].
From the experiment design-time "occlusion-awareness" templates for designers have been created. These templates can be used as an overlay in the design process, to make better occlusion-related layout decisions. Dark blue areas are areas that are occluded with a high probability (>50%), light blue areas are possibly occluded areas (>10%). The findings of the experiment and the occlusion templates can be used in different ways: E.g. the 2 digit transform template can be used to design a rotary dial *(fig. 2(b))*. The icons around the dial can be arranged in such a way that only less important or less frequently used icons are in the occluded area, and more important or often used icons are displayed in the non-occluded area [6].

I think the occlusion problem of the rotary dial could also be avoided by making the icons rotate, while the rotary dial itself is stationary. The rotary dial could have a pin at 12 o'clock (or any other direction) to show which icon is selected. For example, when moving the hand clock-wise, the icons could move counter-clock-wise (and the other way around for moving the hand counter-clock-wise). This way you also don't have to adjust the positioning of the icons dependent of the handedness of the user. A disadvantage of this method would be that you lose the ability to use visual memory. E.g. you can no longer say: "I know I want to use this feature and it's at 9 o'clock". You could counter that, by adding the feature to tap in the middle of the wheel to bring all the icons back to a default rotation, which could also be defined by the user. This way you can still apply your learned knowledge of where which icon/feature is.
The results of Vogel et al. [6] until now were occlusion shapes, that can be helpful for design-time decisions, but it would be better to precisely know the occluded are at any given time. Therefore, the aim was to create a configurable real time occlusion model for future work, similar to the model for pen-based touch input in Vogel's "Occlusion-Aware Interfaces" [4]. Using a diffused illumination tabletop, which captures an image of the hand on/near the surface, Vogel et al. captured the actual hand shape. The arm of the user is not captured, because it is too high. Therefore, a rectangle with a constant offset of 100mm from the centroid has been added, representing the forearm. Vogel et al. extended the previously mentioned model [4] by adding ellipses representing the fingers [6]. The resulting model could be used in the a similar way as the model Vogel et al. created for pen-based input devices [4].

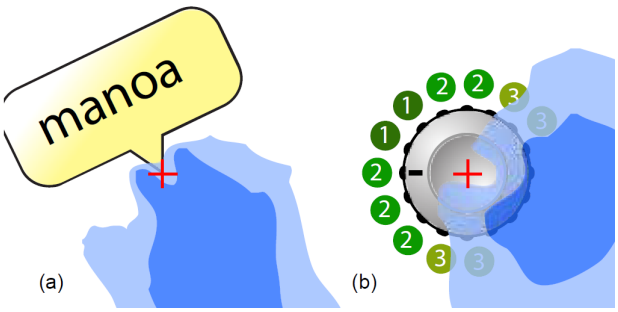

Figure 2. Example usage of Occlusion-Awareness templates

4 Touch Imprecision

Until now we looked at hand-/arm-occlusion. Now we want to have a look at two solutions for touch imprecision and missed feedback caused by finger occlusion.

The Offset-Cursor

One of the simpler solutions to deal with touch imprecision, is the Offset-Cursor [5]. The Offset-Cursor addresses occlusion by creating a crosshair moved by a certain offset from the actual touched location. The user can then move the cursor onto the target and commit his selection by just lifting his finger (take-off selection). This solves the problems of occlusion, touch imprecision, and feedback. But Vogel et al. found significant problems with this method [5]:

Problem 1:
The cursor always appears! You can no longer directly aim for your target. That means, if you wanted to hit a big target, which wouldn't cause any ambiguity, the Offset Cursor would still appear, and you had to move it down to the actual target, your finger was resting on already all the time. This makes easy selection tasks more complicated and take longer. The only way to counteract this behavior, is to compensate for the offset and to try to intentionally hit below the actual target, in hopes that the cursor will directly hit the target above your finger [5]. Vogel et al. [5] showed in an experimental evaluation that the acquisition time using the Offset Cursor for targets, that are large enough to be selected by bare finger, is 1.57 times slower with the Offset-Cursor than without.

Problem 2:
The constant offset to the north makes certain screen areas unreachable, e.g. the bottom of the screen [5].

Problem 3:
"Third, on first use, users are unlikely to expect the offset, aim directly for the actual target and miss." [5]

To address these problems Shift was developed.

5 Shift

The first version of Shift moved the whole screen contents upwards. Users found this to be too distracting, so Shift has been re-designed to use a callout, placed above the finger, instead [5]. When the user touches the screen, Shift copies the occluded area of the screen into a callout, which is presented in a non-occluded area. The callout contains a small crosshair, which the user can move by slightly moving his finger. This way the user can precisely navigate the crosshair into the target and also see the resulting feedback of the app. The selection is committed by lifting the finger of the surface *(fig. 3)*) [5].
Of course it would be a nuisance, if the callout was activated on every target, even if the target was big enough to hit it with no problems. So the callout is only invoked, if the target is small enough and occluded by the user. How soon a callout appears, depends on the size of the target underneath the finger. If the smallest target under the finger is larger than a certain occlusion threshold, then target acquisition is not a problem for the user and the timeout, that defines when the callout appears, can be set to a large value. If the smallest target under the finger is smaller than the occlusion threshold, the timeout is shorter and the callout appears faster to aid the user make his selection. [5]
If the selection is close to the left or right edges, the callout is placed towards the middle of the screen. If the selection is close to the upper edge, the callout is placed to the left, and if that is not possible, to the right.
One thing Vogel et al. did not specify, was how to abort a selection task. What if the callout appears, and I want to abort the selection? One solution I can think of, would be to move the cursor either over empty space in the callout, or if the callout is so crowded, that that's barely possible, the user could be allowed to navigate the crosshair beyond the borders, "outside" of the callout (in the "dead zone") and lift his finger. As soon as the cursor is beyond the borders, the edge of the callout could be highlighted to indicate, that lifting the finger results in aborting the selection task.
The cursor should not be allowed to travel too far into the dead zone, because that could make it hard to bring it back. So the cursor should be allowed to only travel some pixels into the dead zone, and if the user

keeps directing it into the same direction and the cursor is a certain amount of pixels into the dead zone, the cursor stays there and doesn't move any further, despite the user still trying to direct it deeper into the dead zone. This way the cursor can be brought back with minimal movement and the user can quickly try all possibilities, if necessary.

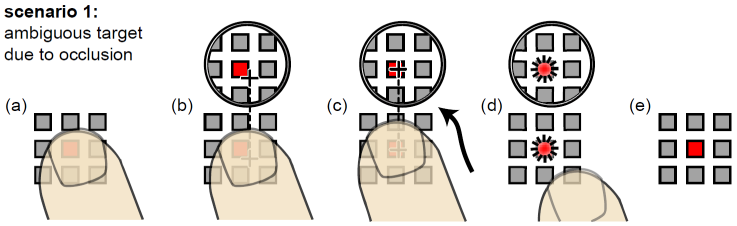

Figure 3. Shift's callout

6 Conclusion

I demonstrated various ways of occlusion, how they affect the user, the perceived feedback, the interaction with the touch screen, and how they can be dealt with. There are different ways of occlusion and different occlusion related problems, like touch-imprecision, and there are different ways to overcome them. You can develop a system, that checks for occluded information, and moves the information to a non-occluded area, but you can or have to also already deal with occlusion in the design process of apps. Because the best way to handle a problem, is to not let it occur at all. So the best way to handle occlusion, is by building apps in a way, that avoids occlusion as often as possible.

REFERENCES

[1] eMarketer. 2 billion consumers worldwide to get smart(phones) by 2016. www.emarketer.com/Article/2-Billion-Consumers-Worldwide-Smartphones-by-2016/1011694, 2015. [Online; accessed 22-June-2015].

[2] eMarketer. Tablet users to surpass 1 billion worldwide in 2015. http://www.emarketer.com/Article/Tablet-Users-Surpass-1-Billion-Worldwide-2015/1011806, 2015. [Online; accessed 22-June-2015].

[3] M. Statistics. Mobile statistics. http://www.mobilestatistics.com/, 2015.

[4] D. Vogel and R. Balakrishnan. Occlusion-aware interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 263–272, New York, NY, USA, 2010. ACM.

[5] D. Vogel and P. Baudisch. Shift: A technique for operating pen-based interfaces using touch. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 657–666, New York, NY, USA, 2007. ACM.

[6] D. Vogel and G. Casiez. Hand occlusion on a multi-touch tabletop. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 2307–2316, New York, NY, USA, 2012. ACM.