Porous Interfaces for Small Screen Multitasking using Finger Identification

Aakar Gupta
University of Toronto
Toronto, Canada
aakar@dgp.toronto.edu

Muhammed Anwar
University of Toronto
Toronto, Canada
muhammed.anwar@mail.utoronto.ca

Ravin Balakrishnan
University of Toronto
Toronto, Canada
ravin@dgp.toronto.edu

ABSTRACT
The lack of dedicated multitasking interface features in smartphones has resulted in users attempting a sequential form of multitasking via frequent app switching. In addition to the obvious temporal cost, it requires physical and cognitive effort which increases multifold as the back and forth switching becomes more frequent. We propose porous interfaces, a paradigm that combines the concept of translucent windows with finger identification to support efficient multitasking on small screens. Porous interfaces enable partially transparent app windows overlaid on top of each other, each of them being accessible simultaneously using a different finger as input. We design porous interfaces to include a broad range of multitasking interactions with and between windows, while ensuring fidelity with the existing smartphone interactions. We develop an end-to-end smartphone interface that demonstrates porous interfaces. In a qualitative study, participants found porous interfaces intuitive, easy, and useful for frequent multitasking scenarios.

Author Keywords
Multitasking; Smartphones; Finger Identification

ACM Classification Keywords
H.5.m. Information interfaces and presentation

INTRODUCTION
Salvucci et al. define multitasking as the ability to integrate, interleave, and perform multiple tasks and/or component subtasks of a larger complex task [25]. Multitasking forms an integral part of the way we interact with a myriad of data on computers with reasonably large screens. Smartphones, with much smaller screens, do not lend themselves naturally to such traditional forms of multitasking. While smartphone operating systems support multiple apps running simultaneously, the limited screen size limits the interface's ability to support multiple concurrently visible and rapidly accessible windows which is the de facto solution for multitasking in larger screen interfaces. The lack of dedicated multitasking interface features has resulted in smartphone users attempting a sequential form of multitasking via frequent app switching. Bohmer et al. [5] describe how users switch repeatedly among already open apps within a short span and how users use groups of apps frequently in sequence. The single window constraint makes this frequent back and forth switching between apps inefficient [19]. In addition to the obvious temporal cost, it requires physical and cognitive effort which increases multifold as the back and forth switching becomes more frequent. Given the numerous obvious mobile form-factor benefits of maintaining a small screen for smartphones, it is clearly worth exploring alternative ways in which to support multitasking on such small screen devices.

Figure 1: Porous Interfaces enable overlaid semi-transparent apps accessible using different fingers as input.

We propose *porous interfaces*, a paradigm to support efficient multitasking on small screens. Porous interfaces enable partially transparent app windows overlaid on top of each other, each of them being accessible simultaneously using a different finger as input. The semi-transparency allows for at least coarse characteristics of data on multiple windows to be discerned concurrently, while finger identification enables concurrent interaction with multiple windows without necessarily bringing the window being interacted with to the top. Further, the interactions enable easy, visible and more fluid data transfer between overlaid windows than is currently possible with traditional "cut-copy-paste" interactions. We designed porous interfaces to

UIST 2016, October 16 - 19, 2016, Tokyo, Japan

ACM 978-1-4503-4189-9/16/10…$15.00
DOI: http://dx.doi.org/10.1145/2984511.2984557

include a set of characteristics that enable a broad range of multitasking interactions with and between windows, while ensuring that the interface maintains existing commonly used and understood interactions of existing smartphone interfaces. We developed an end-to-end demonstration smartphone interface including a hardware prototype that performs finger identification on the touchscreen and a series of applications that showcase porous interfaces. In a qualitative study, participants found porous interfaces easy, intuitive, and very likely to be used regularly if integrated into smartphones.

RELATED WORK

Small Screen Multitasking

Prior research on small screen multitasking is surprisingly sparse. Nagata et al. [22] found that messaging interruptions on a PDA significantly disrupt task performance. Choi *et al.* [6] propose easy notification peeking by flipping the smartphone cover. Mobile device designers have started to recognize the need for easy multitasking on today's mobile devices and have introduced side-by-side windowing on tablets [21]. However, similar solutions are not easily adapted to smartphones owing to their small screen size.

Partially Transparent Windows

While there have been no recent attempts at exploring partially transparent windows in the context of mobile devices, earlier research has explored how they affect content visibility and comprehension on desktops. Harrison et al. [11] introduce transparent layered windows and found that information density in the layers governed the degree of visual distinction between them. Follow up research concluded that image on text or image on image layering is better than text on text layering [12, 13]. Ishak et al. propose content-aware transparency of windows which obscures the unimportant regions in a window [16]. Besides work on layered windows, ToolGlass and MagicLens explore transparent toolboxes on top of an application [4]. However, there have been no published investigations into overlaying full-screen partially transparent windows, and how the user would interact with such an interface.

Finger Identification and other Finger Modalities

In designing porous interfaces, we are faced with the challenge of enabling access to windows layered below the topmost window without brining those windows to the forefront. One possibility is to use a modality such as finger pressure which is already out there in the market. There are two reasons why this would not have worked well. First, we envision the porous interface as an augmentation to the currently ubiquitous smartphone interface. Consequently we need to ensure that all existing gestures supported by an app currently will work in the porous interface seamlessly. With pressure gestures already being used in apps as "force touch", we could not overload the gesture with the new porous interactions. Second, even if we assume that apps using pressure are in the minority, we want seamless multitasking where all gestures including tap, swipe, pinch etc. work well with the overlaid apps. It is difficult to do such gestures with added pressure consistently while switching between less pressure and more at will. Finger Identification allows us to address both these issues.

Prior work has extensively tackled the technical problem of identifying fingers on a touchscreen using fiduciary tags [9, 20], multitouch finger arrangements [1, 7, 29, 30], fingerprinting [8, 14], and muscle sensing [3]. While most have been studied for tabletops, some, notably fingerprinting and muscle sensing, hold promise for future miniaturization. Recent efforts by Apple and others [26, 31] on embedding fingerprinting into displays show commercial interest in this space. The other thread of related work focuses on interactions that use finger identification and is limited in scope, using the same buttons for different actions like cut-copy [3, 10, 20, 24, 28] or using them for chording [7, 9]. Finger-specific chording has been used for multi-step commands [9], mixing brush colors [20], and a virtual mouse [7]. None of the above explorations design end-to-end interfaces based on finger identification. There is a lack of compelling investigation into how these interactions can be uniquely ingrained in interfaces, and how they might lead to better performance. Our work on porous interfaces fills this gap and hopes to fuel the conversation on finger identification interactions.

In summary, no existing works tackle the problem of small screen multitasking in its entirety. We use a novel combination of partially transparent overlaid windows and finger identification on smartphones to design an end-to-end interface for small screen multitasking.

POROUS INTERFACES

As described by Spink et al [27], typical multitasking involves the desire to task switch, the actual task switch, the task execution, and switching back to previous task. We observe this behavior in daily smartphone use in the form of app switching, where a user working on app A desires to switch to app B, invokes the app switcher or goes back to the home screen (1), performs optional swipe(s) to go to the appropriate location on the screen (2), performs a quick visual search to locate app B on the switcher or home screen (3), selects the app (4), waits for the app to open (5), performs tasks in the app, then invokes the app switcher or home screen again to switch back to app A (6), performs an optional swipe again (7), performs a quick visual search again (8), selects app A (9) and waits for it to reopen (10). Excluding task execution in app B, these are a total of 10 temporal, physical, or cognitive steps after the user's desire to switch apps, consuming time, physical, and cognitive effort. While most of these steps are not overly taxing, as users frequently perform app switching, their cumulative effect over time greatly affects user experience and performance. Leiva et al. [19] found that smartphone app switching due to intended and unintended app interruptions may delay completion of a task by up to 4 times.

The situation gets worse when the user has to perform rapid repeated back and forth switching between the apps. Our aim is to reduce this switching time and effort and free users' cognitive resources to work on task related operations rather than window management operations. To this end, we classify three types of operations that the user performs when requiring app switching – switching and viewing the content in app B without interaction, switching and interacting with content in app B, switching and transferring content from app B to app A. The aim consequently becomes enabling the execution of these three operations to happen as rapidly as possible.

Importing the above operations to application windows in smartphones, we delineate three primary prerequisites of porous interfaces that enable efficient multitasking: concurrent visibility of windows, single-step interaction with windows, and instant content transfer between windows. To attain these prerequisites, we define three primary characteristics: transparent overlapped windows for concurrent visibility of windows, different fingers to access different windows in a single step, and multi-identified-finger gestures for instant content transfer between windows. These characteristics are collectively termed as *Porosity*. Another integral property of our system is *Fidelity* which ensures that the porous multitasking interface will not affect the prevailing single window use for touchscreens and will rather be an augmentation to the existing interface.

Porosity

Transparent Overlapped Windows for Concurrent Visibility

Concurrent visibility of windows is the most obvious way to rapidly switch viewing between multiple app. Currently, the way users perform app switching is wholly time multiplexed. Desktops solve the problem by space multiplexing the applications next to each other. This, however, is not possible for small screens. We therefore apply *depth multiplexing* [11], where multiple app windows overlap on top of each other while being partially transparent.

As explored in previous work on transparent windows on desktops, the potential interference of the content of one window with another could be a real problem [12]. We limit the number of overlapped app windows to two to minimize this interference. The two overlapping apps will comprise of a background (or back) app and a foreground (or front) app. The front app is made partially transparent so that both app windows are visible.

Figure 2 (left) shows a user scenario that we implemented. A user is looking at messages from a friend in the front app while looking at their picture gallery in the back app. The information from both apps is coarsely visible. To improve visibility, we use another form of overlap, the *semantically transparent overlap* where the non-useful parts of the front app are made completely transparent so the user can see through those parts. Figure 2 (right) shows the semantically transparent version of Figure 2 (left) which has improved

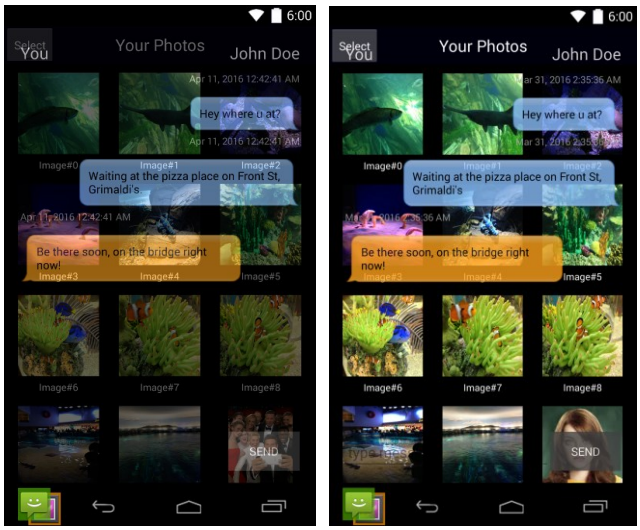


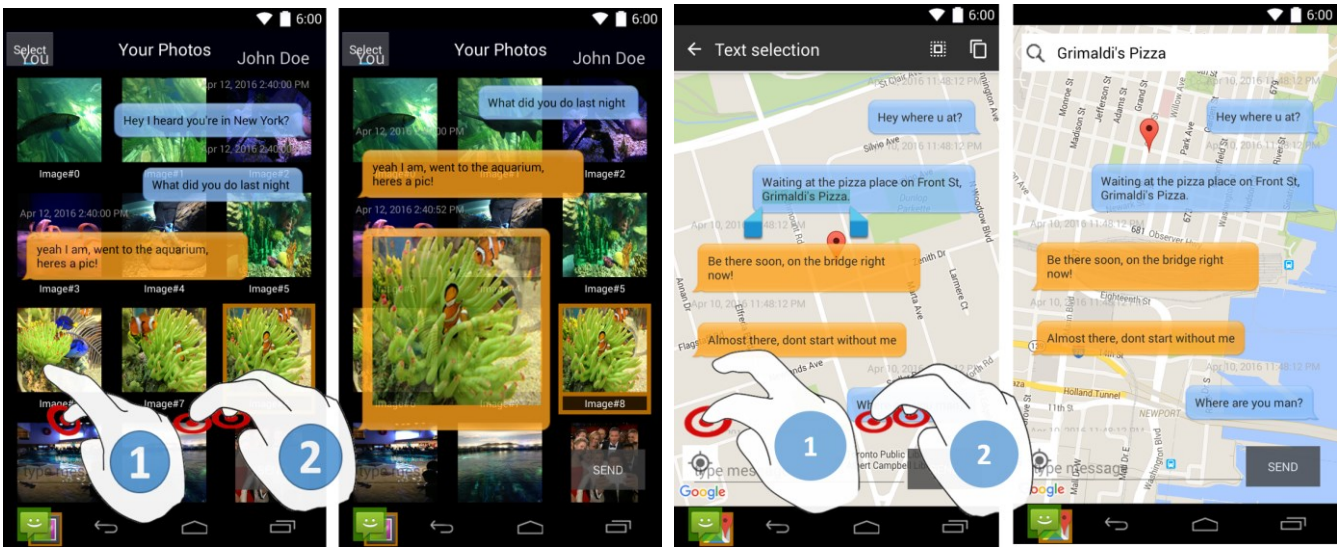
Figure 2: (Left) Messaging on photo gallery, (Right) the semantically transparent overlapped version

visibility. While the semantically transparent apps have been custom-built in our demonstration to showcase the interactions, the concept is an adaptation of Ishak *et al's* content-aware transparency [16] for desktop windows.

Different Fingers for Different Windows

Concurrent visibility of windows poses an immediate problem: if two windows are visible at the same time, then how does the user interact with each window? Prior works address this by keeping only one window interactable at a time. The user can perform a selection command to choose which window to interact with. This mode change approach is suited to desktop interfaces where the windows are only partially overlaid so the user can easily select one or the other by accessing their non-overlaid parts. For small smartphone screens, however, the partial overlay would have to be considerably smaller to make such selections possible. Further, for tasks that require rapid back and forth interactions with both windows, the intermediate selection command upsets the interaction flow and speed significantly, thus making the interaction frustrating. So even though the visibility of windows is depth-multiplexed, the interaction with them is still time-multiplexed. In fact, in a study by Kamba et al. [17] on semi-transparent overlaid widgets over news text, participants overwhelmingly found their way of long press selection of the back layer tedious and requested immediate responsiveness of both layers.

To solve the problem for small touchscreens, we propose different fingers for interacting with different windows at the same time. This reduces the interaction with each concurrently visible window to an immediate single step. In our system, the index finger corresponds to the front app and the middle finger corresponds to the back app. When the two app windows are overlaid, then every interaction on screen with the index finger corresponds to the front app and every interaction on screen with the middle finger corresponds to the back app. For example, if the back app



(a) (b) (c) (d)

Figure 3: Single-step content transfer using the beat gesture. (a), (b) sharing an image from gallery in back to messaging in front using the middle-to-index beat; (c), (d) copy-pasting text from messaging in front to maps in back using the index-to-middle beat.

uses taps, swipes, drags, long press, pressure touch etc. then these operations can be performed using the middle finger. In Figure 2 (b), the user performs messaging with the index finger, and browses the gallery using the middle finger. We will refer to the taps with a specific finger as an index tap or a middle tap and similarly for other gestures such as swipe.

Multi-identified-finger gestures for instant content transfer between apps

The third primary reason a user switches from app A to app B besides viewing or interacting with app B is the need to transfer content between apps. In the smartphone, we can see this manifest in the form of sharing content, getting attachment objects, and copy-pasting content. These operations again take up a lot of interaction steps for the user. We propose a pair of gestures that utilize finger identification to rapidly transfer content between concurrently visible app windows.

To transfer content selected in the front app to the back app, the user index taps the screen (without lifting up), and then middle taps in rapid succession, thus performing a "beat gesture". Multi-finger beat gestures have been shown [23] to operate without interfering with single touch input owing to the duration between the two taps or beats being very small. We augment the beating gesture with finger identification to make an index-to-middle finger beat distinct from a middle-to-index finger beat. Content transfer between the concurrently visible windows is thus made symmetric. Transfer from the front to back app is done using the index-to-middle beat and from the back to front app is done using the middle-to-index beat.

Figure 3 shows two user scenarios that we implemented to demonstrate rapid image sharing and text copy-pasting enabled by the beat gesture. In Figure 3(a), (b), a user messaging in the front app needs to send a picture to her friend from the gallery in the back app. She selects the required picture using a middle long press, and performs the middle-to-index beat to bring it instantly into the messaging textbox, then index taps *Send* to send it. In Figure 3(c), (d) the user is messaging with a friend about a restaurant and needs to look it up in the maps app in the background. She copies the restaurant address and performs an index-to-middle beat to paste it into the maps app's search box.

Based on the above three characteristics, we see how the porous interface simplifies multitasking for small touchscreen devices. For two app windows that are concurrently visible, the number of steps, if the user wants to view the two apps in rapid succession, is reduced from ten to zero. If the user wants to interact with the two apps in rapid succession, the number of steps is reduced from ten to one. If the user wants to transfer selected content, it is reduced to a single multi-finger gesture.

Fidelity

The current interaction paradigm on touchscreens is pervasive and works really well for single app use cases. A new interface paradigm that addresses multitasking should make sure that the existing paradigm is not affected in any disruptive way. We term this property of our porous multitasking interface as the *fidelity* constraint. Fidelity implies two things – that the single app interaction should keep working like before *and* when the apps are overlaid in the porous mode, then all the interactions associated with those apps should be supported.

We will address the first part in detail in the next section. For the second part, we have mentioned how all the single finger interactions associated with both the overlaid apps will be supported by the corresponding finger. The use of the beating gesture was motivated by the fact that it is currently unused in the smartphone interface, besides being

quick and appropriate for our use case. The system also ensures that multitouch interactions such as pinch-and-zoom are supported in the overlaid apps too. When the user wants to zoom into a foreground maps app, she can simply use the standard pinch and zoom gesture using the thumb and the index finger. Similarly a thumb and middle finger pinch and zoom can be used for a background app. However, we recognize that some users prefer to perform pinch and zoom gestures using the index and middle finger. The system currently does not support such a use case. However, a mechanism where the index finger touches slightly earlier than the middle finger to act on the foreground app and vice versa for the background app could alleviate this problem.

Users hold their smartphones in three styles of grips [15]: *a) one handed* – phone is held in the fingers of one hand while interacting using the thumb of the same hand, *b) cradled* – phone is cradled in one hand and tapped with the finger in the other hand, *c) two handed* –phone is held in fingers of both hands with both thumbs providing the input. Since porosity requires two fingers being used in tandem, it can only be used when the user is holding the phone in the cradled style. However, when the second hand is free, the switch from one handed to cradled is easy. We design our system such that existing interactions work in all three usage styles, whether the index finger or thumb is used and porous interface is invoked when the middle finger is used.

AN IMPLEMENTATION OF POROUS INTERFACS

We developed a demo operating system interface as an application within Android that demonstrates porous interfaces and their fidelity with existing interactions.

Finger Identification Prototype

The finger tap is detected by the touchscreen. The prototype then needs to identify the finger whose tap was registered. We explored multiple techniques to identify index and middle fingers distinctly – tracking individual finger movement with optical markers, color markers, and leap motion; capturing differences in finger motion with ringed inertial motion unit (IMU) rings; and muscle sensing. However, none of these approaches worked proficiently within the constraints of our application requirements – a high accuracy for small screens, minimum instances of failure, low communication latency to the phone, and unobtrusive instrumentation. After multiple rounds of experimentation, we settled on our final working design that uses a combined miniature photo-transistor and optical detector sensor mounted on both index and middle fingers.

As shown in Figure 4, the sensor is connected to an Arduino, which processes the sensor data and sends it to a Nexus 4 Android smartphone via a Bluetooth chip. The sensor detects its distance from the touchscreen. It is mounted and calibrated such that when the index finger touches the screen, the distance value is noticeably lower than when the index finger hovers over the screen. We have

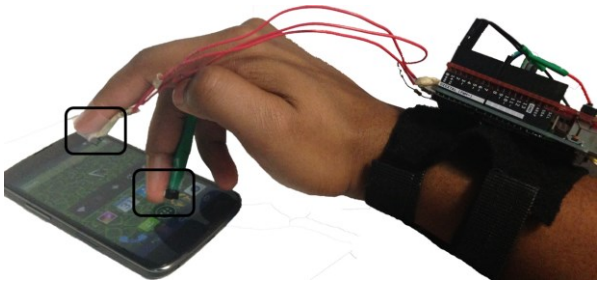


Figure 4: Finger Identification Prototype with IR sensors

a similar sensor on the middle finger. We define thresholds for the distance values for both fingers for when they are touching the screen. When the touchscreen registers a touch, the system checks the distance from both sensors and depending on the threshold, determines which finger was used. If the distance values are high for both the finger, it determines that a finger other than index or middle was used (usually the thumb). An initial pilot with three users showed that upon individual calibration, ~99.5% accuracy was achieved. Each user performed a series of 75 taps and swipes with index, middle, and thumb in random order.

Software System

We built an Android app that simulated an end-to-end smartphone interface that supported porous multitasking. This includes the home screen, notifications bar, window switcher, lock screen, settings features, and nine demo apps. To maintain fidelity, the system should ensure that when the interface is not displaying porous windows, the interaction remains unaffected. The central principle that helps achieve this is *gesture overloading*. When the system is not in the porous mode, user interaction on screen with any other finger besides the middle finger results in the system behaving in the usual way. If the user performs an interaction with the middle finger, for instance, middle tapping an app icon, it results in a response associated with the porous interface. Thus, the middle finger works as a means for implicitly indicating the porous mode to the system.

POROUS WINDOW MANAGEMENT

Kandogan et al. [18] define three processes that impact user performance in a multi-window interface – task window environment set up, environment switching, and task execution. Earlier we described two-window task execution with the window environment already set up. We now look at how porous interfaces enable efficient setup and switching via gesture overloading with the middle finger.

Window Environment Setup

The interface enables invocation of two overlapping app windows while ensuring two things – a) the user can easily designate which app will be the back app and which one will be the front, and b) the single finger invocation of apps should not be affected. A user scenario explains this below.

A user starting a road-trip wants to play songs while constantly looking at their location in the maps app to make

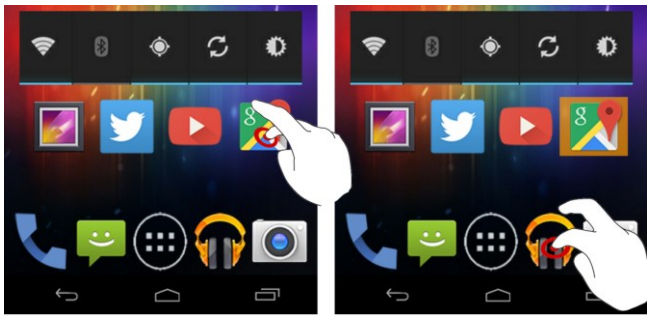


Figure 6: Porous window setup. (Left) Middle tap on app that goes in the back. (Right) Index tap on app that goes in front.

sure they're following the route correctly. The user starts at the home screen as shown in Figure 5 (left). To open Maps+Music, with Maps in the background, the user first middle taps the Maps app which lets the system know that the user intends to open the app in the porous mode and the Maps app is shown to be selected as in Figure 5 (right). The user then index taps the Music app and both app windows are instantly opened with the Music app overlaid over Maps. To open just a single app, the user can use the index finger (or thumb or any other finger besides middle finger) in the usual way. The interface achieves both stated objectives and keeps the interaction limited to two steps.

Since we designed the interface in Android, we overloaded the standard android soft keys *Back* and *App Switcher* as well. When two overlaid windows are open, an index tap on the back button corresponds to its action in the front app and a middle tap corresponds to its action in the back app. If the user continually invokes *Back*, with an index tap for instance, there will come a point when the corresponding front app window will close. At this point, the system will transition out of porous mode. The *Home* button will always take the user to the home screen.

Window Environment Switching

Window environment switching is the act of changing the screen contents to an existing environment setup [18]. In our system, the window environment consists of a pair of apps and their overlay order. We overload the *App Switcher* icon such that when it is middle tapped (Figure 6 (left)), it opens up the porous window switcher (Figure 6 (right)). It shows the thumbnails of the pairs of apps the user is currently working with. Tapping on one of these pairs with either finger opens up the overlaid window pair. The paired app switcher window can also include app pairs that the user frequently works with to enable faster access than the home screen invocation. Index Tapping the window switcher will open the usual single app switcher window where index tapping a thumbnail opens a single app. However, a user might need to invoke a background app while she is working on a single app. For instance, a user messaging with a friend might want to capture a photo instantly to send it. To enable this *on-demand porosity* without a prior setup, all the app thumbnails in the single app switcher window are overloaded such that middle

Figure 5: Paired app switcher. (Left) Middle tap on app switch icon shows (Right) the switcher for pairs of porous windows.

tapping an app thumbnail opens it in the background and the front app becomes partially transparent.

Foreground-Background Indicator

While developing porous interfaces, we sought user feedback at various stages. One feedback was that while users intuitively remembered to use the index finger for the front app and the middle finger for the back app, at times it took them a moment to figure out which app window was at the front and which one at the back even when they had launched the apps in the first place. We included two kinds of feedback to solve this. First, at launch time, the background window was shown instantly on the screen, followed by a delay of 250ms, followed by the animated appearance of the front app such that the animation looked like the front app jumping from above the screen onto the back app. This established an immediate context in the user's mind that the app that appeared from above is the front app corresponding to the index finger. The animation appeared every time the overlaid windows were launched, either from the home screen or from the app switcher. However, if the user does not interact with the two apps for some time, she might forget this association. Therefore, we designed an indicator icon at the bottom of the screen that showed the overlaid app icons in accordance with the window ordering. Figure 7 shows the maps icon on top of the messaging icon. The app icon doubles down as a soft key such that when the icon is tapped with any finger, the ordering of the overlaid windows is toggled which is in turn reflected in the icon. This allows the user to switch their front and back apps at any moment if they so desire.

The default partial transparency of the front window is set at 50% which as noted by prior work works equally as well as a single app [11, 17]. However, depending on the windows' content and the tasks, the user might want to alter transparency themselves. We provide a dynamic transparency control (Figure 7 (left)) which is invoked by

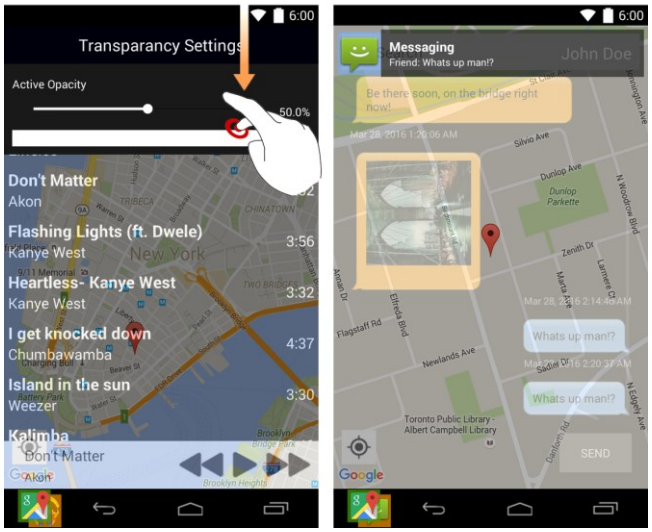


Figure 7: (Left) Dynamic Transparency Control invoked using a middle swipe form top, (Right) the messaging app appears in background when it receives a new message notification

middle swiping from the top bezel. The usual notifications bar appears when using the index or any other finger.

Dynamic Transparency Control

An implicit feature of note here is the *hidden window*. The user can make the front app completely opaque at 0% transparency. In this scenario, the user can still interact with the back app using the middle finger, but can't see it. This would be useful in situations where the user does not want the back app to affect the visibility of the front app, but still wants to interact with the back app. For instance, prior work suggests that when a text window is overlaid on another text window, the visibility of both windows suffers especially if either of the windows is text heavy [12]. A user reading an eBook while playing music in the back app would not like any visual interference but still want to skip songs and replay them using middle swipe gestures that the Music app supports. *Hidden window* enables these kind of user scenarios. Similarly, the transparency can be set at 100% to make the front app the hidden window.

The Vanishing Notification

So far we have talked about situations where the user explicitly intends to multitask and launches and works with overlaid apps. However, there is another kind of situation where the multitasking is not initiated by the user [22]. While performing single app tasks on a mobile device, there are high rates of external interruptions in form of notifications that distract the users from the main task, hamper user performance and delay task completion [19, 22]. While one part of the problem is the cognitive context switch, the second part is the interaction the user needs to perform, especially when the user wants to attend to the notification immediately. For instance, when using app A, the user receives a messaging notification which prompts her to stop the current task, swipe down the notification bar, read the preview and decide to attend to it now, select the notification to launch the messaging app, write a reply, and then perform more steps to get back to her initial app.

We alleviate this problem via the *vanishing notification*. If the user is interacting with a single app and she receives an important notification, the porous mode auto-activates, with the corresponding app window being opened in the background and the front app becoming partially transparent (Figure 7 (right)). The porous mode auto-exits after 3s, the background window disappears, and the front app regains its opacity. The duration is chosen based on prior work which showed that users take a median of 2s to

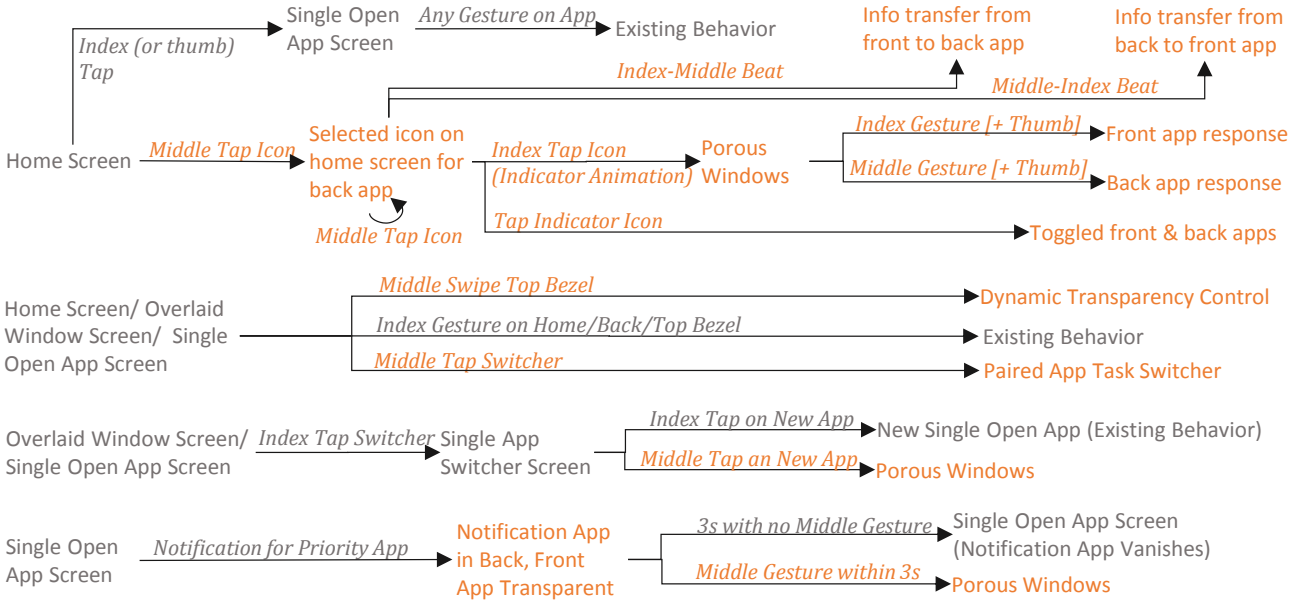


Figure 8: Interaction Flow Diagram for Porous Interfaces which maintain fidelity with the existing smartphone interface. Porous interactions and screens are in orange and existing smartphone interactions and screens are in Grey.

decide if they want to engage with a notification [2]. Since the interface is in porous mode for 3s, the user can interact with the background app using the middle finger. When this happens, the interface stays in the porous mode and does not auto-exit. The user can then use the two windows in porous mode and can switch back to the single window mode by index tapping the app switcher and index tapping the single app or by continually middle tapping *Back*. Since the vanishing notification could feel intrusive to the user, we make it a setting such that it is only allowed for apps that the user explicitly marks as a priority app. Figure 8 summarizes the porous interface interactions and their integration into the existing interface in a flow diagram.

APPLICATIONS

We built a series of demo applications to showcase various user scenarios of how porous interfaces work – Messaging, Photo Gallery, Maps, Music, Video Player, Twitter, Call, News, and Camera. We have already touched upon some of these scenarios such as playing music and navigating on maps during a road-trip, messaging while browsing a photo gallery and messaging a picture from the gallery instantly, messaging notification while reading, and copying text from the messaging app to maps. Other scenarios demonstrated by our apps could include watching a live game while scrolling through live tweets about the game, messaging while watching a video, playing and changing songs without switching out from reading the news, etc. We now discuss applications that show how porous interfaces can lead to even more interesting and optimal extensions.

Camera+Messaging

So far we have discussed how porous interfaces will ease multitasking based on existing app designs. However, if developers can design features specific to porous interfaces in their apps, it could result in unforeseen benefits. We developed a modified camera app (Figure 9), which when overlaid with another app such as messaging, can perform

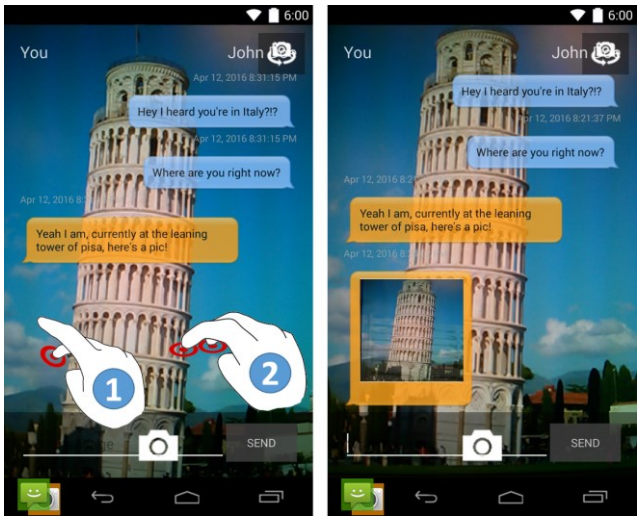


Figure 10: Camera+messaging: the beat gesture captures the picture, and sends it to the chat recipient instantly

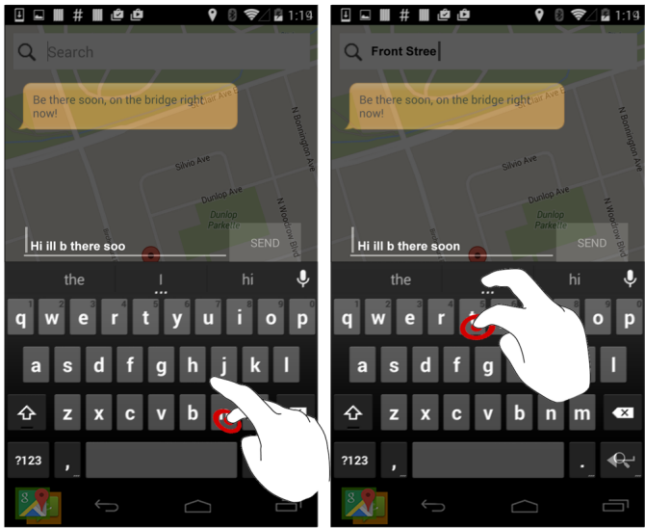


Figure 9: Keyboard works in sync with the porous interface. (Left) Middle tap types in the back messaging app. (Right) Index tap types the address in the front maps app.

photo capture, in addition to sending it to the messaging app, all with the beating gesture alone, thus allowing the user to capture and send images across even more rapidly than a usual camera app overlaid with the messaging app.

Drag and Drop in Droppable Zones

While the beat gesture enabled content transfer between apps, we exploit a property of the Android operating system to enabled drag and drop between two porous windows. In standard Android, when the user starts dragging an object, it automatically detects droppable zones on the window when the object hovers over the zone. With overlaid windows this leads to the following scenario: If the user is browsing a photo gallery in the back app and messaging in the front app, and she long presses an image and starts dragging it, then the image detects the text box on the messaging app as a droppable zone where the user can simply drop the object without using any other gesture. This could be useful in multiple applications such as for email attachments, image sharing between apps, and dragging text items. However, such a technique will not work if the droppable zones of two apps overlap each other at the same location. One solution to this problem that might be implemented in the future is to allow for dynamically movable drop zones that move away from each other across layers so that they never overlap.

Simultaneous Keyboard use in two apps

The keyboard is a system artifact that is usually the same across all apps. In certain scenarios, users perform back and forth typing in two apps in quick succession. For instance, when messaging with a friend about places to eat in the vicinity, the user might want to search maps at the same time, while going back and forth between them. We modify the behavior of the interface such that if both apps contain text boxes, index tapping the keyboard will type in the front

app and middle tapping it will type in the back app, thus reducing the effort even more (Figure 10). The users can stop this behavior via an *off* key on the keyboard.

USER FEEDBACK ON POROUS INTERFACES

We gathered feedback from 8 regular smartphone users (3 female), ages 22 to 30. Our goal was to get user reactions and comments on porous interfaces. The study took 60 minutes per participant. In the first 15 mins of the study, the researcher briefed the participants on the system goals, followed by a demo of porosity interactions, followed by participants playing with each interaction. The next 30 mins consisted of a demo of the window management interactions, and an intro to the 9 demo apps followed by participants playing with each interaction and different app combinations. Participants were free to talk to the researcher, ask questions and comment on their experience. In the last 15 mins, participants were asked to rate the *usefulness* (how useful is a feature) and the *easiness* (how easy it is to perform) for every interaction feature on a Likert scale while assuming a less obtrusive hardware.

Results

Participants uniformly liked porous interfaces and found it intuitive. Seven of eight participants indicated that they are very likely to use the system if integrated into smartphones. Figure 11 shows subjective rankings for each interaction.

Participants liked that they were able to see overlaid apps, however, they had issues with the visibility for some app combinations – *"I want to watch the game and see the twitter stream but when the video gets white, it's hard to see the tweets because their font is white too"*. However, they really liked semantic transparency – *"The chat bubbles are great. I can use them anywhere. It will be great if they can change their color according to my background app."*

Participants loved concurrent access to two apps via different fingers and found finger switching easy but mentioned that they would prefer a less obtrusive hardware setup. A couple of users found a way to interact by tapping the indicator icon to switch app ordering whenever they needed to interact with the back app and kept interacting without changing the finger – *"Usually my other fingers are folded inwards so that I can see the screen completely. I don't want to unclench them, so I just bring the app front."*

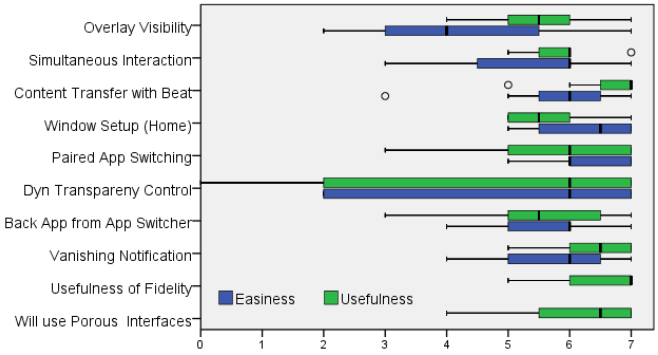


Figure 11: Questionnaire results boxplot.

An anticipated problem was that long nailed users use index and middle fingers to pinch and zoom instead of the thumb, which is not possible in our interface. One participant faced this issue and switched to using her ring finger.

The most uniformly liked were the beat gesture and the vanishing notification. *"I spend all my time attaching files on email or copy pasting things on phone. This copy paste is the killer feature."*, *"It's so fluid. I can just beat pictures to my friends all day."*, *"I don't want to check the notification, but can't stop myself and it stops my work. May be if I see it instantly I can let it vanish without doing anything."* Participants suggested functional extensions – *"The only reason I switch apps is to copy paste or for notification. If the video automatically pauses when I start replying to the notification in background, it would be great!"* One participant had privacy concerns with the vanishing notification – *"So if I look at the chat notification in the background, will it still tell my friend that I read the message? I don't want to tell them immediately every time"*

The most polarizing feature was the dynamic transparency control. Some users liked it. However, others wanted it to be easier and involve only a single step. Participants suggested creative alternatives – "*I don't want to touch the screen to change the screen. Can we use the (hard) volume buttons so that tapping the buttons with middle finger changes transparency?"* We believe the feature is useful, however, it needs a more user friendly execution. An alternative is finger sliding on the bottom soft key row.

Messaging was the most popular app – *"I use chat all the time. I can use it with video, with Facebook, with maps, with news, everything."* In fact, three participants mentioned using video+chat as one of their intended use cases even when we had not explicitly demonstrated them as a pair – *"I can't really focus on the TED talks but if they keep playing in the background while I am chatting, it'll be easier"*. Another participant mentioned using chat with an online shopping browser window – *"It'll be cool to chat with my friend while looking at the products."* The TED comment demonstrates how they could do productive but boring activities while being simultaneously engaged in another app. This desire to do productive things if not for a lack of focus was echoed a lot and they saw a solution in porous interfaces. *"I can play mindless games while reading news…maybe I'll read news more that way."*

Participants also mentioned using video and music in frequent combination with other apps – *"I'll watch Game of Thrones and switch on the camera in the background to record my reaction video!"*, *"I want to play music while recording with my camera so that my personal video will have an automatic background score."* Some more apps that participants mentioned were *Calendar+Email* for scheduling, *Contact List+Calling* for adding friends to an ongoing call without multiple taps, and

Browser+Chat/Email to open links and see them then and there. One participant mentioned an interesting use case for the phone lock screen – *"If I unlock the phone with my thumb or index finger, the phone opens in the mode without any porous interface. If I unlock with the middle finger, it opens the phone with the porous interface."* This is a compelling way to think about the fidelity challenge. If a user intentionally unlocks the phone in the porous interface, she won't expect it to follow the standard interface.

DISCUSSION

Kandogan et al. [18] define five requirements of multi-window systems for multitasking – *i)* allow organization of windows according to tasks, *ii)* allow fast task-switching and resumption, *iii)* free the user to work on tasks rather than window management, *iv)* use screen space efficiently, and *v)* spatially indicate the relationship between windows. Looking back, porous interfaces fulfill all five of these. Porous interfaces, by definition, satisfy *iv),* and enable *i)* and *ii)* using its window setup, app switching, and notification handling. Porous interfaces free the user to work on tasks rather than incessant switching by enabling simultaneous visibility and interaction with frequently paired apps, and rapid content transfer between them, delivering a big improvement over the existing smartphone interface. The indicator animation and icon show the spatial app ordering which is adequate for our paired app use.

However, there are multiple areas for improvement. First, good visibility is conditional upon the windows' contents. Apps that allow for sufficient white space would work well both with apps that allow/don't allow white spaces. Messaging is immensely popular, and allows significant white space. Similarly, apps with item list like music, Twitter, stocks, file browsing, calendar, voice calls, all allow for white spaces. These will work with apps with limited white space like maps, video, news, Facebook, web browser, shopping, photos, email. These may not work well with each other. However, app combinations with an image+text overlay will work well [13] (e.g. photo+email). Semantic transparent overlaps offer a solution to make visibility work more consistently. However, while we custom-built our apps to illustrate the concept, more automated methods need to be explored. A direct way would be to allow developers to indicate segments in their app views which contain less information which can in turn be used automatically by the system to generate porous windows or given to the user to pick and choose the segments they want. Another way is to dedicatedly explore content-aware transparency for overlapped apps on smartphones.

Second, while our system places high emphasis on fidelity, it isn't able to maintain 100% fidelity in all situations as evidenced by the multitouch with long nails issue. Our system is designed to handle a variety of touch input modalities which continue to function as usual, such as pressure. However, if we move to other modalities such as voice and accelerometer motion sensing, the user needs a way to indicate which app should respond to that input. The simplest way to address this would be to always have the front app respond to these, with the user having the control to change the behavior. Another way would be to intelligently predict this based on app behavior and past use. A third solution is to disregard fidelity altogether and design an interface from the ground-up that is rooted in finger identification and transparency. Porous interfaces assume that finger id isn't used within apps. However, both could work together. Porous interfaces only need one additional finger and others can be used for within-app operations. Or, individual apps that use finger id can choose to disable porous multitasking while they are open.

Third, almost all participants said that if the hardware can be compacted into a typical smartring, they will not mind wearing it on their fingers in order to be able to use the interface. However, we recognize that this is not tenable for widespread use and future work will have to focus on developing a ring-free finger identification technique. The point of porous interfaces isn't simply optimizing existing app switching, but opening the space for newer multitasking interactions that result from complete concurrent interactivity of both apps. Consequently, something like explicit mode switching to access different windows is not desirable. As mentioned before, recent commercial efforts could mean that the problem solves itself in the coming years. Plus, misidentified finger handling can be incorporated into the interface by delaying the action to give user the time to undo it when the finger is identified with low surety or blocking high impact operations like email sending in porous mode.

CONCLUSION

In this paper, we proposed porous interfaces that enable small screen multitasking using window transparency and finger identification. To that end, we define their primary characteristics of task execution, the window setup and switching, and helper features. We built an end-to-end interface with fidelity to the existing smartphone interface. We further demonstrated user scenarios using nine demo applications. We gathered detailed user feedback which reinforced the usefulness and ease of porous interfaces. Two instances where users have to switch apps most frequently are for content transfer and attending to notifications. The beat gesture and the vanishing notification directly addressed the two scenarios and received highly enthusiastic feedback from the participants.

Aside from existing scenarios, advanced multitasking on smartphones where people perform complex tasks that require multiple sources of information is infrequent. We need solutions that solve the inefficiencies in existing app switching scenarios *and* that stimulate newer multitasking use-cases and applications for small screen devices. We believe the porous interface is the first step in that direction.

REFERENCES

[1] Au, O.K.-C. and Tai, C.-L. 2010. Multitouch finger registration and its applications. *Proceedings of the 22nd Conference of the Computer-Human Interaction Special Interest Group of Australia on Computer-Human Interaction - OZCHI '10* (New York, New York, USA, Nov. 2010), 41.

[2] Banovic, N., Brant, C., Mankoff, J. and Dey, A. 2014. ProactiveTasks. *Proceedings of the 16th international conference on Human-computer interaction with mobile devices & services - MobileHCI '14* (New York, New York, USA, Sep. 2014), 243–252.

[3] Benko, H., Saponas, T.S., Morris, D. and Tan, D. 2009. Enhancing input on and above the interactive surface with muscle sensing. *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces - ITS '09* (New York, New York, USA, Nov. 2009), 93.

[4] Bier, E.A., Stone, M.C., Pier, K., Buxton, W. and DeRose, T.D. 1993. Toolglass and magic lenses. *Proceedings of the 20th annual conference on Computer graphics and interactive techniques - SIGGRAPH '93* (New York, New York, USA, Sep. 1993), 73–80.

[5] Böhmer, M., Hecht, B., Schöning, J., Krüger, A. and Bauer, G. 2011. Falling asleep with Angry Birds, Facebook and Kindle. *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services - MobileHCI '11* (New York, New York, USA, Aug. 2011), 47.

[6] Choi, K., Song, H., Koh, K., Bok, J. and Jinwook, S. 2016. Peek-a-View: Smartphone Cover Interaction for Multi-tasking. *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '16* (2016).

[7] Ewerling, P., Kulik, A. and Froehlich, B. 2012. Finger and hand detection for multi-touch interfaces based on maximally stable extremal regions. *Proceedings of the 2012 ACM international conference on Interactive tabletops and surfaces - ITS '12* (New York, New York, USA, Nov. 2012), 173.

[8] Fitzpatrick, G.P., Haynes, T.R. and Williams, M.L. 1995. Method and apparatus for accessing touch screen desktop objects via fingerprint recognition. 1995.

[9] Goguey, A., Casiez, G., Vogel, D., Chevalier, F., Pietrzak, T. and Roussel, N. 2014. A three-step interaction pattern for improving discoverability in finger identification techniques. *Proceedings of the adjunct publication of the 27th annual ACM symposium on User interface software and technology - UIST'14 Adjunct* (New York, New York, USA, Oct. 2014), 33–34.

[10] Gupta, A. and Balakrishnan, R. 2016. DualKey: Miniature Screen Text Entry via Finger Identification. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '16* (San Jose, California, USA, 2016).

[11] Harrison, B.L., Ishii, H., Vicente, K.J. and Buxton, W.A.S. 1995. Transparent layered user interfaces. *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '95* (New York, New York, USA, May 1995), 317–324.

[12] Harrison, B.L., Kurtenbach, G. and Vicente, K.J. 1995. An experimental evaluation of transparent user interface tools and information content. *Proceedings of the 8th annual ACM symposium on User interface and software technology - UIST '95* (New York, New York, USA, Dec. 1995), 81–90.

[13] Harrison, B.L. and Vicente, K.J. 1996. An experimental evaluation of transparent menu usage. *Proceedings of the SIGCHI conference on Human factors in computing systems common ground - CHI '96* (New York, New York, USA, Apr. 1996), 391–398.

[14] Holz, C. and Baudisch, P. 2013. Fiberio. *Proceedings of the 26th annual ACM symposium on User interface software and technology - UIST '13* (New York, New York, USA, Oct. 2013), 41–50.

[15] Hoober, S. 2013. How do users really hold mobile devices. *UXmatters, http://www. uxmatters. com/mt/archives/ ...*. (2013).

[16] Ishak, E.W. and Feiner, S.K. 2004. Interacting with hidden content using content-aware free-space transparency. *Proceedings of the 17th annual ACM symposium on User interface software and technology - UIST '04* (New York, New York, USA, Oct. 2004), 189.

[17] Kamba, T., Elson, S.A., Harpold, T., Stamper, T. and Sukaviriya, P. 1996. Using small screen space more efficiently. *Proceedings of the SIGCHI conference on Human factors in computing systems common ground - CHI '96* (New York, New York, USA, Apr. 1996), 383–390.

[18] Kandogan, E. and Shneiderman, B. 1997. Elastic Windows. *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '97* (New York, New York, USA, Mar. 1997), 250–257.

[19] Leiva, L., Böhmer, M., Gehring, S. and Krüger, A. 2012. Back to the app. *Proceedings of the 14th*

international conference on Human-computer interaction with mobile devices and services - MobileHCI '12 (New York, New York, USA, Sep. 2012), 291.

[20] Marquardt, N., Kiemer, J., Ledo, D., Boring, S. and Greenberg, S. 2011. Designing user-, hand-, and handpart-aware tabletop interactions with the TouchID toolkit. *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces - ITS '11* (New York, New York, USA, Nov. 2011), 21.

[21] Multitasking on Mobile Devices: 2015. *https://www.nngroup.com/articles/multitasking-mobile/*. Accessed: 2016-03-21.

[22] Nagata, S.F. 2003. Multitasking and Interruptions during Mobile Web Tasks. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*. 47, 11 (Oct. 2003), 1341–1345.

[23] Oakley, I., Lee, D., Islam, M.R. and Esteves, A. 2015. Beats. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI '15* (New York, New York, USA, Apr. 2015), 1237–1246.

[24] Roy, Q., Guiard, Y., Bailly, G., Lecolinet, É. and Rioul, O. 2015. Glass+Skin: An Empirical Evaluation of the Added Value of Finger Identification to Basic Single-Touch Interaction on Touch Screens. *INTERACT: IFIP International Conference on Human-Computer Interaction* (2015).

[25] Salvucci, D., Kushleyeva, Y. and Lee, F. 2004. Toward an ACT-R General Executive for Human Multitasking. *ICCM*. (2004).

[26] Sonavation has bonded 3D fingerprint sensors to Gorilla Glass: 2015. *http://thenextweb.com/insider/2015/07/21/sonovation-has-bonded-3d-fingerprint-sensors-to-gorilla-glass-kiss-your-home-button-goodbye/*. Accessed: 2015-09-11.

[27] Spink, A., Cole, C. and Waller, M. 2009. Multitasking behavior. *Annual Review of Information Science and Technology*. 42, 1 (Nov. 2009), 93–118.

[28] Sugiura, A. and Koseki, Y. 1998. A user interface using fingerprint recognition. *Proceedings of the 11th annual ACM symposium on User interface software and technology - UIST '98* (New York, New York, USA, Nov. 1998), 71–79.

[29] Wagner, J., Lecolinet, E. and Selker, T. 2014. Multi-finger chords for hand-held tablets. *Proceedings of the 32nd annual ACM conference on Human factors in computing systems - CHI '14* (New York, New York, USA, Apr. 2014), 2883–2892.

[30] Westerman, W.C. and Elias, J.G. 2006. System and method for packing multi-touch gestures onto a hand. 2006.

[31] Yousefpor, M. and Bussat, J. 2014. Fingerprint Sensor in an Electronic Device. *US Patent App. 14/451,076*. (2014).