Multi-User Smartphone-Based Interaction with Large High-Resolution Displays

Lynn Nguyen and Jürgen P. Schulze

University of California San Diego, Calit2, La Jolla, CA

ABSTRACT

We investigate the practicality of using smartphones to interact with large high-resolution displays, such as tiled display walls. To accomplish such a task, we found that in most cases it is not necessary to find the spatial location of the phone relative to the display, rather we can identify the object a user wants to interact with through image recognition. The interaction with the object itself can be done by using the smartphone as the medium. This trivially allows multi-user interaction with a large display wall, provided that each user has a smartphone. To investigate the feasibility of this concept we implemented a prototype.

Keywords: smartphone, tiled display wall, user interaction

1. INTRODUCTION

Large high-resolution displays, such as tiled wall displays, are a common visualization tool in research laboratories and are becoming commonplace to the general public such as in presentations or conference halls. The visualization laboratory at the California Institute for Telecommunications and Information Technology (Calit2) has multiple tiled display wall installations. One of them is the AESOP Wall, which consists of a 4×4 array of thin-bezel 46" monitors from NEC (see Figure 1).

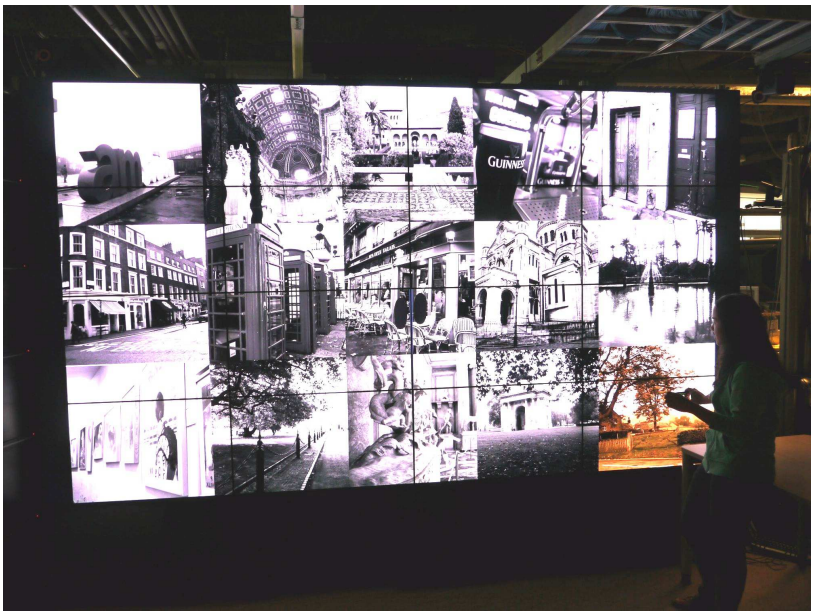

Figure 1. (Video 1): Tiled display wall with a collection of photographs and user with smartphone, selecting an image. http://dx.doi.org/doi.number.goes.here

While these displays are great for displaying data, they can be difficult to interact with because of their size. Common methods of interaction include using a desktop control station with a standard USB mouse, a

Further author information: E-mail: {lkn001|jschulze}@ucsd.edu

freehand presentation mouse (air mouse), or a tracking system.[1] Unfortunately, these methods are not without their shortcomings. Using a control station with a mouse is bothersome because for every pixel on the control station that the cursor moves, it may map to moving tens of pixels on the high-resolution display; this approach also represents a physical constraint on the user to the station. An air mouse allows the user to move around but is still unable to account for the physical location of the user. Because the cursor position of the mouse is independent of the location of the user, it is easy to lose track of the cursor. Furthermore, in order to move the mouse from one end of the screen to the other, the user would have to ratchet the mouse to the correct position instead of just being able to point the mouse at a desired location. A 3D tracking system is able to address the issues presented by using a mouse, but the tracking system can be even more expensive than the display itself. Another issue is that these displays tend to have multiple viewers at once. However, with many of the current navigation methods in place, there is only one person (the driver) who controls the display while everyone else depends on the driver to navigate the data set. Typically, these methods do not promote a multi-user environment and can be described as expensive as well as unintuitive.

Moreover, these methods assume the user wants a precise location on the display. However, this level of granularity is not required of all applications. It may be possible to treat the data on the display as objects. In this way, we can think of interaction with the display in two parts: at a course grain level where the user needs to determine the object on the display to interact with, and at fine grain level where the interaction is localized to the object. Consider this concept in practice with the following scenario. Imagine you are a research scientist exploring new territory with your camera. After a day out in the field, you come back to the lab to analyze the pictures with your colleagues. A convenient way to do so would be to present the images collected on a large wall display while conversing and taking notes about each image in turn. The course grain interaction is determining which image to focus on while the fine grain interaction is the note taking on the selected image. With many colleagues, this becomes a time consuming task because everyone must move at the rate of the driver. A prospective solution to this issue is to leverage technology available through today's smartphones in order to create a rich experience for the scenario just described.

Smartphones are an attractive alternative because they have many features, are budget-friendly, and available almost everywhere. In fact, sales for mobile communication devices have shown an increase in 19 percent from the first quarter of 2011, and in particular smart phones have experienced an increase of 23.6 percent.[2] These numbers are going to continue in an upward trend due to the ever increasing feature set provided by these devices. Many cellphones today come equipped with a GPS unit, camera, and wireless internet capabilities, among others. For the purposes of discussion, the term smartphone used in this paper will mean a mobile device equipped with a touch screen and at least Wi-Fi and a camera. The availability, feature set, and price make smartphones an attractive alternative device to interact with large displays.

In this paper, we investigate the practicality of using smartphones to interact with large high resolution displays. One of the ways the prototype we developed addresses the shortcoming present with traditional input methods is by completely eliminating the need for a precise cursor location on the display. Also, our prototype easily supports multi-user interaction while staying within a small budget to create an intuitive work environment for large high-resolution displays.

2. RELATED WORK

Using camera equipped cellphones to interact with large displays is not a new concept. A popular technique for such a configuration is to track the mobile device using software markers. An example of this technique is demonstrated through the Point & Shoot method.[3] The phone display is used to aim the camera at an image on a display. When the selection button is pressed, a visual code flashes briefly on the display to determine the selection coordinates. While accurate in regards to the selection, the brief flash of the visual code seems as though it would be distracting to other users of the system. Additionally, the user needs to concentrate on the display of the camera rather than the large display to align the visual codes. Many software toolkits, such as SpotCode[4] and ARToolkit,[5] are available now that make this functionality available to users. It is not necessary to show the visual code on the display, but instead the visual code may be printed on a piece of paper in the user's environment. However, this is inconvenient as the user is burdened with setting up this extra infrastructure.

Direct Pointer[6] is a system that does not rely on the use of visual codes. It utilizes the camera and internet capability on a mobile device to control the cursor on the display. In order to move the cursor, the cursor on the display must be captured in the frame of the camera on the mobile device. Then, the device may move and the new cursor position can be calculated by the view frames being sent to the server. This system has the advantage of not having to deal with visual codes, but has the same disadvantage of that of an air mouse the user may still lose track of the cursor on a large display.

ARC-Pad[7] utilizes the touchscreen on mobile phones to control cursor location. The touchscreen is then a portable trackpad that allows a user to select an absolute location on the large display by double tapping on the corresponding location of the touchscreen and a relative location can be determined by sliding the finger. This technique allows the user to easily find the cursor and does not obstruct the user's display with visual codes. However, to reach a position on a wall sized display that is not on the edge requires a significant amount of finger sliding. For very large displays, this is still not as intuitive an approach as just pointing the phone to a location on the display. Additionally, using the touchscreen of the mobile device as a trackpad makes it a very limited possibility of taking advantage of the screen to display relevant information.

Takacs et al.[8] presented a system which recognizes real objects using the camera of a smart phone. Our image recognition approach is similar to theirs, but our contribution is not an image recognition algorithm, but the multi-user environment of a large display wall at which multiple users can work independently with a shared image data base.

We published initial results of this project as a poster at SIGGRAPH 2012,[9] but it did not include the detailed technical description or analysis we provide in this article.

3. SYSTEM DESCRIPTION

3.1 Hardware

Our experiments were run using an HTC Aria Android phone without a data plan. Original query frame resolution is 768x432. Library images were captured by several different camera models. Unfortunately, none of the cameras were equipped with built-in GPS units, so GPS metadata was added to the images manually via Picasa to demonstrate the power of using a smartphone as an interaction device. The library images were displayed on the AESOP wall located in Calit2.

3.2 System Design

For many uses of large displays, fine grain movement is not necessary. If this is the case, we do not need to know the exact pixel coordinate of the users cursor. We can then use computer vision algorithms to match what the phone is seeing to an image in the library to determine what the user is pointing at.

The system is essentially a client-server model communicating via TCP/IP (see Figure 2. The server is outfitted with a database to store user comments on images.

The client is an Android device with integrated camera and wireless internet connection enabled. It will continuously send image frames (query images) of what its camera sees at a rate of two frames per second. From time to time, the client will also send comments on an image for the server to store in the database. The client does not do feature extraction since it would be more efficient on the server side. The only processing the client does is converting the query frame to JPEG and scaling it to a size the server requests. Scaling is done on the client side to reduce the amount of data being sent across the network.

The server receives query images from the client and extracts features based on the SURF[10] algorithm using the OpenCV library. Once these features are extracted, a matching image is determined using a nearest neighbor algorithm provided by the Fast Library for Approximate Nearest Neighbors (FLANN).

The main advantage of such a system is that it does not rely on additional infrastructure such as a tracking system or visual code.

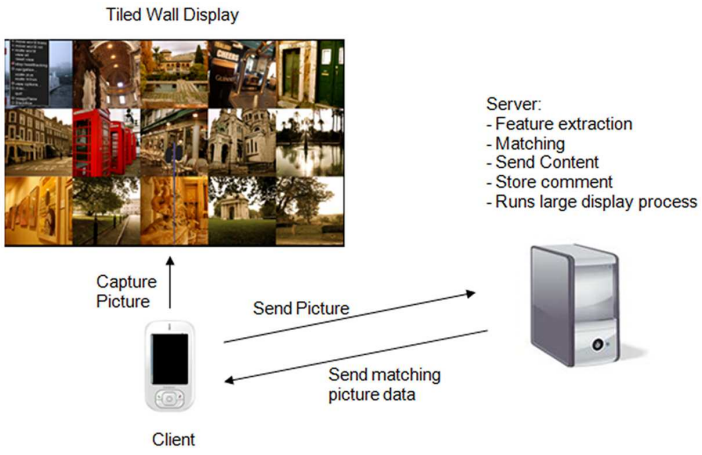

Figure 2. High-level architecture of prototype system.

3.3 User Interface

When running the client application on the phone, the user initially sees that the phone acts as though it is in camera mode. The user is able to use the screen as a view finder. As the user directs the view finder at different images on the wall, if the system is able to match the image that the phone is viewing with one in the library, then image metadata (such as date taken, camera make taken with, focal length and shutter speed of the camera) is overlaid on top of the camera preview in the top left corner. Additionally, the selected image on the screen will be highlighted. Once this text appears in the corner (and the corresponding image is highlighted on the display), the user is then able to select an image by tapping on the screen. On tap, a menu will appear at the bottom of the screen (see top picture of Figure 3).

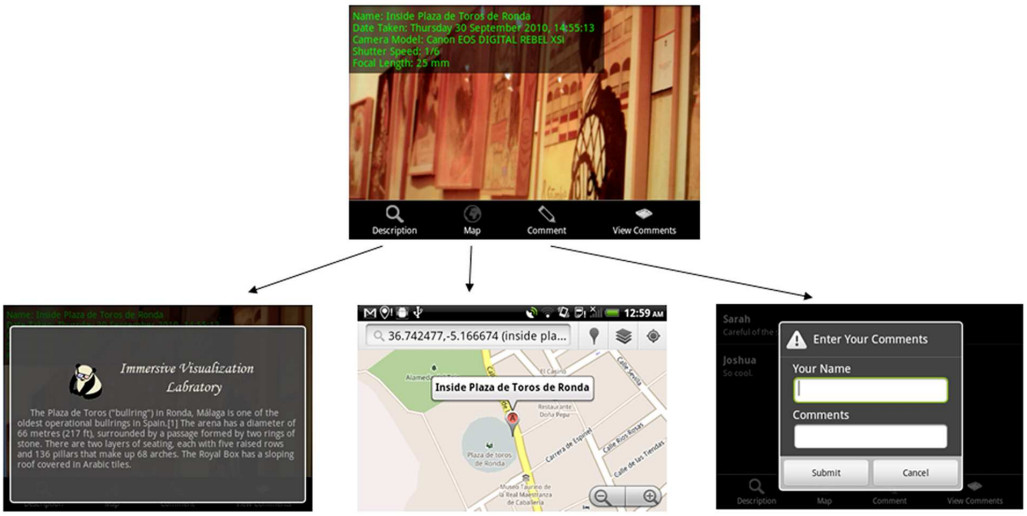

Figure 3. Menu Options.

Once the menu opens, the user is allowed to choose from four options. The user can view a description of the image, see on a map the location where the image was taken (only if the image metadata contained GPS data), as well as view or make comments on the image selected. To exit out of the menu and resume scanning to select other images, the user merely taps outside of the menu.

4. IMPLEMENTATION

4.1 Image Recognition

Image recognition is accomplished by matching what the cellphone sees to an image in the reference library. This is done using standard vision techniques to extract features from the query image and finding matching library images using a nearest neighbor search.

In the backend, the user specifies in a configuration setting the path to the directory that holds the library images. A python script is then run to create and populate a database with metadata scraped from the library images (the metadata pulled is defined by the Exif (Exchangeable image file format) image standard; we use a subset of it, see Figure 4). When the server runs, the server loads the images and extracts the features to store into memory for matching with query images.

```
TABLE imagedata (
        id INTEGER PRIMARY KEY,
        name TEXT,
        path TEXT,
        description TEXT,
        camera_make TEXT,
        camera_model TEXT,
        datetime TEXT,
        shutter_speed TEXT,
        focal_length TEXT,
        gps_lat REAL,
        gps_long REAL
)
```

Figure 4. Schema for image metadata.

4.1.1 SURF Feature Extraction

If the query images sent from the client to the server were perfect cropped versions of the original library image, then it would have been easy to use template matching to determine the exact location on the screen that the cellphone was focused on. However, this would rarely be the case because the user will hold the camera at odd angles or may stand far or close to an image. Taking this into account, an algorithm was needed that would be able to handle transformed images. SURF fit the bill since it is a rotation- and scale-invariant algorithm. Additionally, SURF was reported to be much faster than other algorithms, such as SIFT.[11] This was in the interest of the project because the system needed real-time matching in order to provide the user with a seamless experience. Using SURF, we can determine what image the user is pointing at without needing to know the exact coordinates of the cellphone's view.

Features were extracted from images using OpenCV's SURF algorithm. Through experimentation the system saw more accurate matches when each library image only had 300-500 descriptors. Instead of programmatically keeping the top 300-500 descriptors, we tweaked the parameters until we were about to get the target number of descriptors. For the prototype implementation, we used a Hessian threshold of 2000. This, of course, will vary between different types of images. In addition, the system only uses basic SURF descriptors (64 elements each) as opposed to extended descriptors with 128 elements. Further attempts to reduce false matches was to only match query images that had more than 50 descriptors.

Experimentation consisted of using the cellphone to take photos of the library images from different angles. Then the algorithm was run and the match percentage was determined. For extra visual verification during the development process, a result image was outputted in which the query image was displayed above the library image and lines were drawn between matching points on the images (see Figure 5).

4.1.2 Matching Using Nearest Neighbor Search

In order to obtain the actual matched image, we need to compare the query image against every single library image. The comparison is a nearest neighbor algorithm provided by FLANN. For each library image, a match percentage is determined by the number of matched descriptors in the query image over the total number of

Figure 5. Visualization of matching descriptors made by the server.

descriptors in the query image. The highest match percentage obtained determines the library image that the query image is matched to. In order to reduce the number of false matches, this percentage must be higher than a pre-determined threshold (obtained through experimentation), then the matching library image ID is returned.

4.2 Network

4.2.1 Data Transfer

All communication is done over a wireless network using TCP/IP. The server only sends strings to the client. The client on the other hand sends either strings (when the user makes comments on an image) or JPEG images (when sending query frames for image matching) encoded as bytes to the server. Since the data varies, it is necessary to set up a header packet for data sent from the client to the server. The header consists of 5 bytes. The first 4 bytes make up a an integer in big endian format that describes the size of the data being sent. The 5th byte in the header is a character indicating the command for the server to know what to do with the data it received. Depending on the command, the server will either run the matching algorithm, or insert a comment into the database for a library image.

C++ and Java interaction is tricky. Strings sent from the Java client to C++ server needed to be converted to ASCII and manually null-terminated before sending across the network. Strings sent from the C++ server to the Java client needed to be manually terminated with a newline character because the client side uses InputStreamReader to read data sent from the server.

4.2.2 The Life of a Query Image

On the client side, frames are captured and sent to the server at a rate of two frames per second. The data captured is stored as a bitmap byte array in YUV format encoded as NV21.[12] This data is then converted to RGB format, compressed to a JPEG format, and scaled to the size that the server expects. When the server obtains the JPEG bytes, it decodes the bytes into an IPLImage, which is the format that OpenCV uses to handle images.

4.2.3 Making Comments

Once the client is able to recognize an image through the matching algorithm, the user is able to make a comment on the image. The client sends the comment to the server along with the image ID and the name of the user. The server takes the data and stores it in a database. The comments are a many-to-one relationship, and we model this using a traditional relational database as a table that references the image data table (see Figure 6).

```
CREATE TABLE comment (
        id INTEGER PRIMARY KEY,
        image_id INTEGER,
        commenter TEXT,
        blurb TEXT,
        FOREIGN KEY(image_id)
            REFERENCES imagedata(id)
)
```

Figure 6. Schema for user comments.

4.3 Large Display Interaction

In order to showcase the ability to interact with the images with the large display, feedback for image selection is enabled. That is, when a user selects an image by tapping on the phone screen, the corresponding image (if any) is highlighted on the wall display.

Currently, the graphics display and the server are running as two separate processes. In order for the server to communicate to the graphics process which image is currently selected, the image ID is stored in shared memory. The image ID is just a 32 bit data type (integer) that is stored by the server and read by the graphics process before rendering each frame. The graphics process can then determine which images are to be highlighted based on what is read from the shared memory block.

To expound upon a point raised earlier, it is easy to support multiple users. The users can view information about the data on the wall at their own pace without needing to wait on the driver. Additionally, highlighting the image upon user selection is a cue to other users that someone is working on an image.

4.4 Exploiting Smartphone Features

The ability to use the display on the camera as a preview for what the mobile device is sending to the server is already taking advantage of features available to almost all of today's smartphones. Additionally, some phones, such as the Android ones, contain a native map application. The Exif standard allows for GPS location metadata tags. It is easy to extract this data and allow the phone to use the native map application to plot the GPS coordinates (see bottom middle picture in Figure 3).

5. RESULTS

5.1 Phone Position

The matching algorithm is relatively accurate. For the best match results, the image should fill the majority of the camera's field of view, especially if multiple images are involved. This could mean that only 1/10th of the image could be seen through the phone screen, just as long as that 1/10th of the image fills up the majority of the phone display. False matches were very rare, and sometimes no image would be matched if the phone was in between two images on the display.

5.2 Demo Results

As a reference point, the latency between seeing an image on the screen and seeing a match is no more than a second for a library of 15 images. When running just the matching algorithm with the same parameters, the average time per match was 0.35 seconds.

5.3 Parameters for Matching

The match time and accuracy is very dependent on the parameters chosen for the matching process. The user can adjust the Hessian threshold of the SURF feature extraction. A good parameter for 12Mpixel images is 2000. It is not recommended to use extended descriptors instead of basic descriptors. Through trial and error, we have found that using extended descriptors doubles the time for matching as well as decrease the accuracy. Finding decent parameters was actually one of the more difficult items we came across because there is no good

documentation about how to choose parameters. In the end, we observed that it is based off of the number of descriptors one can extract from a picture. Our goal was a combination of scaling (down) images and adjusting the Hessian threshold until we were able to get an average of 300 descriptors.

time to match (s) - using 1000 hessian threshold

library image size (px)	query image size (px) 160x120	320x240
300x300	0.2115	0.3926
400x400	0.1739	0.4

time to match (s) - using 2000 hessian threshold

library image size (px)	query image size (px) 160x120	320x240
300x300	0.1409	0.2656
400x400	0.174	0.3586

% wrong - using 1000 hessian threshold

library image size (px)	query image size (px) 160x120	320x240
300x300	0.2	0.16
400x400	0.35	0.19

% wrong - using 2000 hessian threshold

library image size (px)	query image size (px) 160x120	320x240
300x300	0.4	0.2629
400x400	0.4	0.1935

Figure 7. Values obtained from averaging the results of 65 query images against a library of 15 images.

The data provided by Figure 7 show that varying the Hessian threshold parameter when extracting features according to the SURF algorithm has an affect on the matching algorithm. We can see that decreasing the Hessian threshold leads to a lower percentage of false matches while increasing the amount of time it takes to match as compared to the higher value Hessian threshold. It takes longer to match with a lower Hessian threshold because SURF will extract more features. Also, we can see that decreasing the number of pixels to compare leads to faster match times but at the cost of less accurate matches.

6. DISCUSSION

In retrospect, it would have been more interesting to programmatically grab the top 100 or 300 descriptors ranked by the Hessian value to see if setting the Hessian value or adjusting the image size really has any affect on accuracy.

There are many benefits to using a smartphone as an interaction device for a large display. The many features of the phone allow the user to select objects on the display in an intuitive manner while offering the user many more functions than just selecting. Using the phone allows users to see more data independently of a driver and allows multiple users to view the same screen without hindering each other. Also, the phone has other features such as Wi-Fi capability which enables users to collaborate with each other at the same display wall, as well as access powerful smart phone applications, such as Google Maps for data visualization.

7. LIMITATIONS

One thing to note is that duplicate images that the user wants to treat as distinct entities will not be treated correctly by the system. This could become an issue in the future if someone wanted to use the system to interact with a large wall display and there were duplicate images that they wanted to see side-by-side as the adjusted brightness and contrast.

Currently the time it takes for the system to match a query image to a library image scales linearly with the number of images in the library. As the system gets bigger, this will be unwieldy. However, there are many techniques such as parallelism and more sophisticated matching algorithms that may render this to a non-issue.

8. CONCLUSION

From the prototype developed, it can be seen that it is feasible to distinguish between course and fine granularity when interacting with large displays. Smartphones can be used to achieve this end by using computer vision techniques to recognize objects on the display, and the capabilities of the phone itself to have further control of a recognized object.

9. FUTURE WORK

Additionally, the system could broaden the set of functionality the smartphone provides once an object is recognized. For example, more fine grain control such as object manipulation is a possibility. In this way, we could build expand upon using the smart-phone as a interactive controller for a large wall display.

More research would need to be done to disambiguate the case when there are duplicate or very similar objects on the display. A possible suggestion would be to use neighboring image information or motion of the device.

ACKNOWLEDGMENTS

We would like to thank Professor William Griswold for loaning us the HTC Aria smartphone and providing us with feedback in the initial stages.

REFERENCES

[1] Thelen, S., "Advanced Visualization and Interaction Techniques for Large High-Resolution Displays," in [*Visualization of Large and Unstructured Data Sets - Applications in Geospatial Planning, Modeling and Engineering (IRTG 1131 Workshop)*], Middel, A., Scheler, I., and Hagen, H., eds., *OpenAccess Series in Informatics (OASIcs)* **19**, 73–81, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2011).

[2] Gartner, "Press releases. may 19, 2011." `http://www.gartner.com/it/page.jsp?id=1689814`.

[3] Ballagas, R., Rohs, M., and Sheridan, J. G., "Sweep and point and shoot: phonecam-based interactions for large public displays," in [*CHI '05 extended abstracts on Human factors in computing systems*], *CHI EA '05*, 1200–1203, ACM, New York, NY, USA (2005).

[4] Madhavapeddy, A., Scott, D., Sharp, R., and Upton, E., "The spotcode project website, http://www.cl.cam.ac.uk/research/srg/netos/uid/spotcode.html," (2012).

[5] "Artoolkit website." `http://www.hitl.washington.edu` (2012).

[6] Jiang, H., Ofek, E., Moraveji, N., and Shi, Y., "Direct pointer: direct manipulation for large-display interaction using handheld cameras," in [*Proceedings of the SIGCHI conference on Human Factors in computing systems*], *CHI '06*, 1107–1110, ACM, New York, NY, USA (2006).

[7] McCallum, D. C. and Irani, P., "Arc-pad: absolute+relative cursor positioning for large displays with a mobile touchscreen," in [*Proceedings of the 22nd annual ACM symposium on User interface software and technology*], *UIST '09*, 153–156, ACM, New York, NY, USA (2009).

[8] Takacs, G., Chandrasekhar, V., Gelfand, N., Xiong, Y., Chen, W.-C., Bismpigiannis, T., Grzeszczuk, R., Pulli, K., and Girod, B., "Outdoors augmented reality on mobile phone using loxel-based visual feature organization," in [*Proceedings of the 1st ACM international conference on Multimedia information retrieval*], *MIR '08*, 427–434, ACM, New York, NY, USA (2008).

[9] Nguyen, L. and Schulze, J. P., "Image based smartphone interaction with large high resolution displays," in [*ACM SIGGRAPH 2012 Posters*], *SIGGRAPH '12*, 71:1–71:1, ACM, New York, NY, USA (2012).

[10] Bay, H., Ess, A., Tuytelaars, T., and Van Gool, L., "Speeded-up robust features (surf)," *Comput. Vis. Image Underst.* **110**, 346–359 (June 2008).

[11] Lowe, D. G., "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision* **60**, 91–110 (Nov. 2004).

[12] "Android developer documents." `http://developer.android.com/reference/android/graphics/ImageFormat.html`.