Making Mobile Augmented Reality A Reality

Laurent Bindschaedler
Ecole Polytechnique Fédérale de Lausanne
EPFL IC, Station 14
CH-1015 Lausanne, Switzerland
laurent.bindschaedler@epfl.ch

Hendrik Knoche
Ecole Polytechnique Fédérale de Lausanne
EPFL IC, Station 14
CH-1015 Lausanne, Switzerland
hendrik.knoche@epfl.ch

Jeffrey Huang
Ecole Polytechnique Fédérale de Lausanne
EPFL IC, Station 14
CH-1015 Lausanne, Switzerland
jeffrey.huang@epfl.ch

MobileHCI 2011, Aug 30–Sept 2, 2011, Stockholm, Sweden.
ACM 978-1-4503-0541-9/11/08-09.

Abstract
Recent advances in mobile device technology have freed augmented reality (AR) applications from the constraints of desktops, laptops and head-mounted displays. But they are met with a lack of guidelines on the design and user interactions of mobile device-based AR systems. The situation for developers is further exacerbated by closed-license environments and inflexible solutions. We provide an overview of the design of AR applications on handheld devices, the necessary building blocks and problems that future AR systems need to overcome. This experience was gathered during the design and development of an AR framework for the Android™ platform. User experience evaluations showed a great demand for overlay collision avoidance and the value of being able to freeze AR screens. These will be valuable for the design of future mobile device based AR applications.

Keywords
Augmented reality, Design, Engine, Evaluation.

ACM Classification Keywords
H5.1. Information interfaces and presentation (e.g., HCI): Multimedia Information Systems – Artificial, augmented and virtual realities.

General Terms
Design, Human Factors.

Introduction

Augmented reality (AR) applications typically infuse a live video capture of the real world with value-added information. A view of reality is thus enhanced with digital media such as graphics, text or sound called overlays. In AR systems, both real and virtual worlds coexist, allowing the user to interact with the context [1]. Typical applications of AR target museums, navigation, military cockpits, sightseeing, advertisement and education [2].

However, actual implementations on personal use are few and far between, mostly due to the complexity and costs involved in designing and implementing such systems. Faced with an absence of practical software development support, we designed an Augmented Reality Engine (ARE), which provides a proof-of-concept of AR applications on current hard- and software platforms and the basis for an AR application framework on the Android™ platform.

The next section briefly surveys the existing state of the art for AR. We then introduce and discuss the design of the ARE and conclude with some preliminary insights into the design and User eXperience (UX) of AR applications on handheld devices.

Background

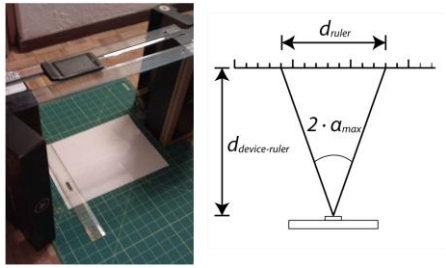
Most of the 1990s AR research proposed augmented reality platforms based on head-mounted displays and handheld devices. Now, with sufficient hardware in processing, cameras and sensors, smartphones and slate tablets have become an option for discretionary applications. Two distinct AR application paradigms have emerged: location-based and vision-based. Both are based on drawing overlays onto a picture or video. In location-based AR, overlays are drawn on the screen based on the location of data items, e.g. pointing the camera towards the Eiffel tower would show historical information. On the other hand, vision-based AR leverages computer vision techniques to recognize and track objects, e.g. drawing a virtual teapot on a real table. The most prominent location-based AR applications on mobile devices are Layar [3] and Wikitude [4], which provide some application programming interfaces (APIs). They tend, however, to relegate developers to data providers as they do not allow for much user interface customization and providers must be validated in a sandbox approach.

Currently, the main vision-based AR tools are the ARToolKit [5] and the Qualcomm AR Software Development Kit [6]. They provide basic computer vision algorithms but more advanced functionalities, such as object tracking and distance computation, are often not available. Nevertheless the status quo in AR development hinders the creation of novel AR applications and stymies the exploration of this field.

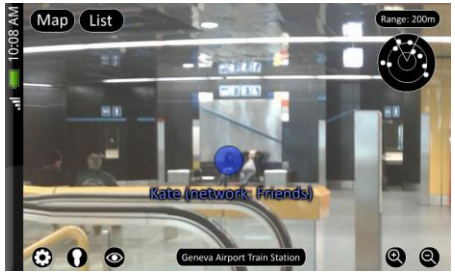
Augmented Reality Engine

Our ARE simplifies building location-based AR applications. Initially, it was designed as an Adobe Flash™ prototype for desktop and smartphones. Once the prototype evaluations were satisfactory, we implemented the ARE on the Android™ platform.

The ARE supports a *standalone* and an *embedded* mode within another application. In *standalone* mode, AR views can be easily deployed by providing the ARE with the information to display, e.g., to display adjacent restaurants. The ARE can also be used in *embedded* mode, where the developer has much finer control, e.g. to implement viewing restaurant menus, ratings and making reservations.



Makeshift experimental setting to measure camera viewing angles — We take a picture of a ruler from a known distance in both horizontal and vertical directions and then measure the span of the ruler on each picture. The view angles are computed using trigonometry. This typically results in a less than 1% error.



Screenshot of the first Flash™ prototype.

The high-level software architecture of the ARE consists of three subsystems. The display subsystem handles the camera rendering, the display of overlays and all touch-based interaction. The positioning subsystem manages the sensors (compass, accelerometer) and the Global Positioning System (GPS) unit and adapts the frequency settings for sensor updates. The data subsystem manages different data sources representing, for instance, a person or a location.

Location-based AR applications need to obtain the location and orientation of the device, as well as the location of data items. The orientation is defined by means of three angles (azimuth, roll and pitch) provided by sensors. Raw orientation values tend however to oscillate (due to shaky hands or sensor inaccuracy) and result in jittery or jerky overlays. Similarly, the location updates of data items can be sparse or inaccurate.

In order to prevent related display issues, it is a good idea to interpolate the movement of overlays on the screen. The ARE interpolates the angle values by sampling angle values using an Exponentially Weighted Moving Average (EWMA) scheme as follows:

$$\propto_i = k \cdot \propto_{measured,t} + (1 - k) \cdot \propto_{i-1}$$

The value of *k* (typically 0.1) is empirically determined from the frame rate to provide a smooth experience. To avoid aliasing problems, we always interpolate across the smallest angle in absolute value and adjust the sampling frequency of angles so that the difference between successive samples is less than 180°.

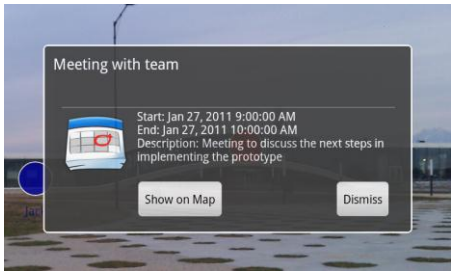
Given the device orientation and location, the screen position where an overlay should be drawn can be computed based its location and altitude. We first obtain relative horizontal and vertical angles and map these to screen positions by a proportionality rule. Proper mapping of overlays to on-screen positions requires calibration of the camera view angles. In the margin, we describe a fast technique to measure these angles using basic trigonometry and a ruler.

The display subsystem is the main processing bottleneck. We used two tricks to speed it up. First, we optimize the selection process of overlays to draw on the screen based the field of vision. Overlays are sorted by distance and filtered by proximity to the device before they are tested for containment within a circle enclosing the screen area. A special caching technique is used where filtered overlays are marked as being outside the field of view. The cache is invalidated on device movement or change of orientation. Second, we optimize the view tree. The straightforward approach of drawing each overlay on its own layer, with superimposed layers, complicates the view tree immeasurably. Instead, all overlays are drawn in a single layer, which however requires the ARE to handle the drawing of overlays and dispatch of events (e.g. clicks) to the appropriate overlays.

Innovative features

The ARE supports a variety of customizations for overlays. The programmer can define their own shapes and colors for each overlay (2D or 3D). The ARE supports non-location based overlays that require an on-screen position, e.g. an arrow to guide the user. Interaction with overlays, such as defining behaviors for click and long click, is fully customizable. Display dialogs, drag and drop and customized haptic feedback can be based on this. The engine can display text along with the overlays or in dedicated text boxes. Finally, overlays can be drawn with adaptive translucency.

Example of an ARE application.



Display dialog associated with an overlay — Text under the overlay is shortened, while it is displayed entirely in the dialog. Text shortening can help reduce overlap between overlays.

We provide a freeze frame feature for users to pause the camera preview and the overlays, so they can examine the situation in a more relaxing posture. It is similar to taking a picture with the camera but the overlays remain fully interactive. In addition, users can zoom on it with a pinch. Unlike the camera preview, the frozen frame has the maximum camera resolution.

UX Evaluation

We used and evaluated the ARE with an AR application that augmented existing calendar solutions and location-based social networking by displaying, e.g., geographical location of friends using Google Latitude. The prototyping was interleaved with expert reviews and UX evaluations were conducted on task-based scenarios with four young adults, mostly students. We obtained feedback both during the evaluation (through talking aloud protocols) and after (debrief interviews).

One of our early findings related to the complexity of the user interface was the need for a lean interface, which did not include too many indicators or buttons. We found that some users enjoyed the sense of empowerment provided by having all the features readily available, but most were confused and wary of clicking on the wrong button. As a result, we strived to provide an uncluttered interface where functionality does not get in the way of the main AR experience.

Throughout our evaluations, users were very concerned about collisions, almost as if the overlays represented cars about to collide and found it "*very distracting to see overlays occupy the same point in space*". Interpolation can further intensify collisions, especially in case of rapid movements. Limiting the number of overlays and making use of depth information to adjust the size of overlays can help reduce collisions.

Great attention was paid to making the overlays and the interaction with the application as smooth as possible, in order to convey a sense of control to the user. The freeze frame feature received nothing but positive feedback. Quoting one user, this feature "*helps diminish the feeling of being stupid standing in a crowd, looking like you're taking pictures*".

UX evaluations also confirmed the fact that a seamless and continuous integration with other applications on the device (such as calendar, contacts, messaging, etc.) was necessary to solve common tasks.

Conclusion

We presented the Augmented Reality Engine, an AR library which illustrates the concept of Do It Yourself Augmented Reality. Besides introducing the reader to the design of location-based AR applications, we shared some UX findings from our various evaluations, which emphasized the need for seamlessness. We also introduced some novel UX improvements and a new interaction paradigm that allows freezing the view.

References and Citations

[1] Azuma, R. A Survey of Augmented Reality, *Presence: Teleoperators and Virtual Environments*, August 1997.

[2] Kostaras, N. and Xenos, M. Assessing the usability of augmented reality systems. *13th Panhellenic Conference on Informatics*, September 2009, 197-201.

[3] Layar Reality Browser website. http://www.layar.com

[4] Wikitude website. http://www.wikitude.org

[5] ARToolKit website. http://www.hitl.washington.edu/artoolkit/

[6] Qualcomm Augmented Reality SDK website. http://developer.qualcomm.com/dev/augmented-reality