Operating System Support for Augmented Reality Applications

Loris D'Antoni[1], Alan Dunn[2], Suman Jana[2], Tadayoshi Kohno[3], Benjamin Livshits[4], David Molnar[4], Alexander Moshchuk[4], Eyal Ofek[4], Franziska Roesner[3], Scott Saponas[4], Margus Veanes[4], Helen J. Wang[4]

[1]University of Pennsylvania [2]University of Texas at Austin
[3]University of Washington [4]Microsoft Research

Abstract

Augmented reality (AR) takes natural user input (NUI), such as gestures, voice, and eye gaze, and produces digital visual overlays on top of reality seen by a user. Today, multiple shipping AR applications exist, most notably titles for the Microsoft Kinect and smartphone applications such as Layar, Wikitude, and Junaio. Despite this activity, little attention has been paid to operating system support for AR applications. Instead, each AR application today does its own sensing and rendering, with the help of user-level libraries like OpenCV or the Microsoft Kinect SDK.

In this paper, we explore how operating systems should evolve to support AR applications. Because AR applications work with fundamentally new inputs and outputs, an OS that supports AR applications needs to re-think the input and display abstractions exposed to applications. Unlike mouse and keyboard, which form explicit, separate channels for user input, NUI requires continuous sensing of the real-world environment, which often has sensitive data mixed with user input. Hence, the OS input abstractions must ensure that user privacy is not violated, and the OS must provide a fine-grained permission system for access to recognized objects like a user's face and skeleton. In addition, because visual outputs of AR applications mix real-world and virtual objects, the synthetic window abstraction in traditional GUIs is no longer viable, and OSes must rethink the display abstractions and their management. We discuss research directions for solving these and other issues and building an OS that let multiple applications share one (augmented) reality.

1 Introduction

The last few years have seen the arrival of consumer *augmented reality* (AR) applications. An AR application takes natural user interactions (such as gestures, voice, and eye gaze) as input and overlays digital content on top of the real world. For example, on mobile phones, augmented reality "browsers" such as Layar and Junaio allow users to look through the phone and see annotations about a magazine article or a storefront. Furniture applications on the iPad allow users to preview what a couch would look like in the context of a real room before buying [13]. The Kinect has sold over 19 million units and allows application developers to overlay avatars on top of a user's pose, creating new kinds of games and natural user interfaces. Even heads-up displays, previously restricted to academic and limited military/industrial use, look set to reach consumers with Google's announcement of Google Glass [21]. The area has also received sustained academic attention, which we discuss in Section 4.

Today's operating systems provide no special support for AR applications. As a result, today's AR applications are built as one-off experiences, where the application itself performs sensing, rendering, and user input interpretation (e.g., for gestures), aided by user-space libraries such as the Kinect SDK or OpenCV. Today, these applications can only run one at a time, and they receive *unrestricted*, *exclusive*, access to read the input sensors and render augmentations. However, this approach faces two major challenges, *user privacy* and *lack of support for concurrent applications*, which we discuss next.

First, it is undesirable to give any application complete access to video and other sensor streams. Consider Figure 1, which shows a video frame captured from a Kinect using the Kinect for Windows SDK. In the current model, any application using the SDK has access to the raw video and depth stream. In this case, that includes the user's face, the contents of the whiteboard, and a bottle of medicine.

Therefore, we must rethink how the OS interprets and delivers user input to applications while preserving user privacy. The OS needs a finer-grained

Figure 1: Video frame captured from a Kinect in the office of one author. The picture contains multiple pieces of sensitive information: the face of the author, drawings on the whiteboard, and a bottle of medicine with the label showing.

and more usable permission system to allow least-privilege permission granting in the AR environment. For example, rather than simply granting an application access to the camera, as is done on today's smartphone systems, the OS needs to control access to the user's face, skeleton, eye gaze, wall, etc.

Second, rather than continue running AR applications one at a time, we argue that it is desirable and compelling to let multiple AR applications from different vendors *simultaneously* read sensor inputs and render virtual overlays in a *shared* 3D reality. Evidence is mounting that such an multi-application AR platform can be highly desirable. For example, mobile phone "AR browsers", such as Layar, have created APIs for third parties to write applications on top of Layar. Currently, Layar sports over five thousand applications in its "Layar Catalogue"[16], each of which adds different annotations to the world.

We argue that the necessary support for such apps should be integrated into the OS. Rather than unnecessarily duplicating heavyweight image processing logic such as face or skeleton detection in each AR application, such logic would be shifted into a new, centralized OS module, simplifying AR application development and letting the OS efficiently manage performance and battery life.

When we argue for integration into the operating system, we do not mean that functionality must necessarily be placed in the OS kernel. Instead, we mean that *shared, trusted* functionality that is *isolated* from each untrusted application should be provided. We have a prototype "AR OS" implementation, for example, where shared object recognition functionality is placed in a single userspace Windows 8 process and multiplexed over local sockets.

We recognize that new abstractions need justification given Engler and Kaashoek's jeremiad against "one size fits all" OS abstractions[9]. The key difference we see is that our abstractions encapsulate security and privacy between AR applications and so cannot be provided by untrusted applications. For example, we cannot give applications low-level access to camera hardware to do their own gesture recognition and at the same time hope to prevent those applications from learning sensitive data from raw video. There may be techniques to cut this Gordian knot, such as the use of restricted, functionally pure languages to implement an OS extension [15].

Overall, AR applications require new mechanisms and abstractions from the operating system: (1) multiplexing sensor inputs and recognized input gestures across multiple applications, (2) protecting those inputs with a fine-grained access control policy enforced centrally by the OS, and (3) providing presentation abstractions for applications to render realistic augmentations in a shared reality. Such support will face new challenges not seen in traditional OSes, such as noisy user input interpretation, fine-grained permission granting, managing non-rectangular, realistic 3D objects from mutually distructing applications, and physics-enabled cross-application interactions.

2 AR Overview

What is an AR Application? For the purposes of this paper, an augmented reality application (1) recognizes the presence of objects or events in the world by applying machine learning to raw sensor data, such as gestures by a person or a landmark. Then the application (2) outputs a set of "virtual objects", optionally attaching them to recognized real-world objects, and renders them in real time. For example, the Facebook "live poster" application in Section 1 recognizes the presence of a wall, then outputs a virtual "poster" with Facebook status updates that is rendered to look like it is on the wall.

Form Factors. We expect AR systems to have three major form factors, two of which are shipping today to consumers. First, *phone AR* consists of a phone or tablet acting as a "magic window" on an augmented world, such as Layar. Second, *room AR* consists of a fixed set of cameras in a room feeding into a large display, such as the Microsoft Kinect. Finally, *wearable AR* refers to augmentation through glasses, body worn camera, or other always-on worn device, such as Google Glass. While other form factors, such as AR clothes kiosks or AR billboards, will exist, these three form factors are the starting

Application	Objects Recognized
Kinect Applications	
Your Shape 2012	skeleton, person texture
Dance Central 3	skeleton, person texture
Nike+Kinect	skeleton, person texture
Just Dance 4	skeleton, video clip
NBA 2K13	voice commands
Dashboard	pointer, voice commands
Forward-looking Applications	
Personal Assistant	face, full audio text
FB Live Tile	wall positions
Furniture	room positions
Translation	Text OCR

Figure 2: Analysis of sample applications to determine how often "raw" sensor access is needed.

points for multi-application user platforms.

These form factors have different input and output characteristics that affect the privacy issues for each. With room AR, the camera faces the user. This raises privacy issues for the user, because in room AR the user is continuously observed. Visual feedback is limited to a single screen. Inputs are noisy, consisting primarily of voice and gestures. The sensors, however, are limited to a single room and do not travel.

With wearable AR, cameras typically face outward, away from the user. These devices are mobile, so video feeds implicitly leak location information and identities of people around the user. While the user may be able to selectively point a camera by turning his or her head, this is not common. Input comes through gestures and voice, which also have noise. Output may be immersive, depending on the type of visual and audio features used.

In contrast, phone AR falls in between. The phone may have two cameras. When using the camera facing out, it sees other people and so has privacy issues similar to wearable AR. When using the camera facing in, it has privacy issues similar to room AR. Compared to the other form factors, the phone is explicitly controlled by the user, with a default of not showing video to any applications.

3 Sharing One Reality

We start with aspects of AR that make it different from desktop or mobile phone application models. For each we discuss similarities and differences with other systems, such as proposed ubiquitous computing systems.

From 2D Rectangles to 3D Meshes. Desktop and mobile phone systems typically display application content through rectangular windows. These windows are arranged by the OS and are static. In contrast, AR applications create objects defined by three-dimensional triangular meshes and position them in a 3D space. Therefore, the OS needs to expose 3D objects as the display abstraction and perform display isolation and management in the 3D space. One particularly interesting aspect is that real-world 3D space is difficult to virtualize. If a region is occupied by one application's object, then another application cannot show its object in the same view at the same time. Overcoming this problem may require creating new visual metaphors for objects, enforcing behavior consistent with these metaphors in the OS, and educating users what these metaphors are. This aspect is specific to AR.

Mixed Real and Virtual Objects. Traditional desktop and mobile phone applications do not typically need to position windows with regard to objects in the real world. In contrast, AR applications create virtual objects and must update their position as the user's view of the real world changes. This imposes real time requirements beyond what is currently found in today's GUI sysems. The AR objects, to look realistic, may obey physics and as a result will move on their own or as a result of interacting with other objects. The OS needs to handle the physics between objects of different applications (such as a ball of one application bouncing into a ball of another application), which can be considered a new kind of cross-application interaction. The OS also needs to deliver a richer set of events like "collision" events in addition to traditional events like user input or network data arrival events.

Continuous, Noisy Input. The AR systems we consider do not have a mouse or keyboard. Instead, inputs are continuous, driven by gestures, objects in the environment, and speech from the user. In addition to being continuous, these inputs are *noisy*: the algorithms for mapping raw sensor data to a user input have false positives and false negatives. For example, the Kinect may recognize a chair as a skeleton and hence return junk skeleton positions to an application. This is unlike desktop systems, where keyboard and mouse are relatively reliable. Mobile phones are somewhere in between, as touch keyboards already face this issue of correcting for user mis-types. This aspect applies to any system using "natural user input," but again AR systems will be a prominent class of such systems.

3.1 Research Problems and Directions.

Continuous input in tension with privacy. The continuous input nature of AR systems means that

applications need access to video, audio, and other sensors for control. The key problem is that, as we saw in Figure 1, these sensors can include sensitive private data, including the face of the user, the contents of the whiteboard, and a bottle of medicine. Unfortunately, we cannot turn off these sensors entirely, because otherwise we cannot control the application!

The research problem here is how the OS should manage natural user input is presented to applications and input is in band with other sensed data. For example, the OS might create abstractions that expose only the data required to applications. This would include a standard vocabulary of gestures (such as "swipe", "wave", or "grab") and other input abstractions. As a first step toward estimating the feasibility of such a standard vocabulary across multiple applications, we analyze the top 5 best-selling Kinect-enabled XBox applications, along with the XBox Dashboard. For each application, we enumerate the objects they recognize. Figure 2 shows the results. Surprisingly, the only application that uses the raw video feed is a dancing game, Just Dance 4. This game allows users to optionally take short video clips of themselves dancing and send these videos to others. The other applications instead simply render an avatar on top of the user's skeleton position. One application, NBA2K13, uses only voice commands and no video data at all.

We continue with four"forward-looking" applications. The Personal Assistant recognizes faces and keywords in conversations, then lets the user know the name of a person and what to say. The Facebook Live Tile, as we discussed in Section 1, renders Facebook status updates as a virtual "poster" on the wall of a room. The Furniture application places virtual furniture into spaces of a real room. Finally, the translation application uses OCR and then adds "subtitles," as in the shipping Bing Translator application on phones. Again, none need the full video stream. The personal assistant optionally uses the full audio stream to assist in person recognition.

These preliminary results indicate that an OS which prohibits continuous access to "raw" video and audio, yet exposes higher-level gestures, could suppport today's AR applications. Another tool in our toolbox might be *access control gadgets,* (ACGs) which are special system UI elements that mediate access to privileged resources [23]. One challenge here is that AR applications have long term, continuous access to sensitive resources such as video data. A second challenge is that AR applications will need fine-grained permissions, such as access to faces in a video stream, while ACGs are defined for coarse-grained resources such as access to the camera. That said, the "send a short video clip" behavior seen in Just Dance 4 might be implemented with a system-controlled ACG. The application would ask the system to pop up a dialogue and the application would supply a destination for the video, but the application would never see the raw video. More work remains to see if this approach would work for additional AR applications, as well as working through the impact on OS permissions, but this initial analysis suggests that "least privilege for AR" at the OS level is possible.

Performance and battery life. Because AR applications render on top of a user's senses, the platform has strong real-time requirements to avoid distracting lag between the real world and virtual objects, possibly as low as 7 ms drawing latency [1]. Continuous object recognition is also processor intensive. We found that one algorithm for computing room geometry from Kinect depth data could not run at acceptable frame rates on anything less than an Nvidia GeForce 650GTX, which has a gigabyte of RAM and 384 cores. The research question is how to intelligently mix local and remote computation while minimizing bandwidth usage and preserving user privacy. While offloading is a classic problem, the latency requirements and object recognition needed in AR make it particularly challenging.

Input ambiguity. The OS must deal with the inherent noisiness of machine learning for recognizing gestures, speech, and other input. While of course we hope these recognizers will be perfect, today object recognition algorithms generally have false positives and false negatives. In addition, because new object recognition algorithms arrive all the time, the OS must also tackle how third paty applications could extend the platform with additional object recognizers. There are two key research challenges here. First, how can an OS deal with inherently probabilistic input? Second, how can we "sandbox" third party object recognizers so they do not interfere with each other or leak sensitive information against the wishes of the user?

A key scenario here is analogous to"clickjacking"' [12]: the user interacts with the system, but unwittingly ends up communicating with the "wrong" application. Because input is continuous and noisy, this can happen even by accidennt. For example, if two applications register to be notified on two words that sound similar, the system may become confused about which application to notify. Therefore a research challenge for this example is designing an OS that protects the user against malicious applications

that intentionally register for confusing words so as to gain improper access to user information.

Application Space Management. As applications move from windows to three-dimensional objects, the OS must manage how they present to the user. The most natural approach is for the OS to centralize presentation across all applications. Because AR applications create 3D objects, this would incorporate a physics and graphics engine like Unity. A central trusted renderer enables the OS to make guarantees about performance, distraction, and app interference. Applications, however, may want to extend presentation with new ways of rendering. Addressing this tension may require the OS to open up new channels from applications to the GPU that can strongly isolate rendering in different parts of a display.

Distraction. Depending on the output device, there may be patterns that will distract the user or even cause discomfort. For example, flickering at certain frequencies will trigger seizures in epileptics, and in fact a specially targeted animated GIF has been used to attack epileptic users of a web forum [18]. More prosaically, an application could detect an oncoming car, then put up a virtual object that shields the car from he user's view. A system with a central trusted presentation stage has an ideal choke point for addressing this issue, because then the system can prioritize the user's attention based on application priority. The research challenge here is first discovering these patterns, then determining ways to avoid them without sacrificing expressivity or speed.

Object Model. Another challenge is how the OS should mediate between applications that are "embodied" in virtual objects. This aspect has similarities to work on virtual reality, where applications are similarly embodied. We expect problems seen in VR environments, such as the self-replicating "grey goo" objects in Second Life [14] to also appear in AR. As a result, AR systems will need to implement mechanisms from VR, such as object creation rate limiting [17]. What complicates AR is that, unlike VR, virtual objects must "interact" with real objects that cannot be controlled or inspected by the system.

Application Semantics. Finally, there may be benefits from a restricted programming model that makes it easier for the system to reason about application behavior. We have made an initial foray here with a functional language focused on simple annotation of recognized objects. Because the language is not Turing-complete, it supports precise static analyses, including an analysis that detects potential annotation conflicts between applications [6].

4 Related Work

Azuna surveyed augmented reality, defining it as real-time registration of 3-D overlays on the real world [3], later broadened to include audio and other senses [4]. We take a broader view and also consider systems that take input from the world. Qualcomm now has an SDK for augmented reality that includes features such as marker-based tracking for mobile phones [22].

Shipping object recognizers include the Kinect skeleton detection algorithm [24]. Another common recognizer is face detection [25]. More recently, Poh et al. showed that heart rate can be extracted from RGB video [20].

Our notion of taking raw sensor data and providing a higher level abstraction is similar to CondOS [5]. We differ in that we focus on recognition of real-world objects. Using static analysis to guarantee non-interference is similar to Singularity's use of managed code to run multiple programs in the same address space [2]. We noted that mobile systems may need to offload object recognition to the cloud or to specialized hardware, which raises heterogeneity issues similar to those tackled by the Helios satellite kernel architecture [19]. The issues in allowing individual applications to extend a trusted renderer are similar to those faced by GPU virtualization [7].

A common approach to privacy for sensed data is to add noise to the data. Differential privacy is a definition of privacy that, if met, gives strong guarantees against an adversary learning about any specific individual [8]. Unfortunately, in our case, adding noise to the raw data would likely lead to errors in recognized inputs. A permission based approach is more promising, and recent work focuses on the psychology of asking for permissions in Android and other systems [10, 11]. In our setting, however, this work does not directly apply because AR is immersive and has fine-grained permissions such as access to all faces in a video stream but not other objects. More work is needed to explore a permission experience for AR applications.

5 Conclusion

We have introduced multi-application augmented reality as an emerging issue for systems. These systems raise new issues that prompt new directions in OS research. The time is right for the systems community to consider multi-application augmented reality.

References

[1] Michael Abrash. Latency - the sine qua non of ar and vr, 2012. http://blogs.valvesoftware.com/abrash/latency-the-sine-qua-non-of-ar-and-vr/.

[2] Mark Aiken, Manuel Fähndrich, Chris Hawblitzel, Galen Hunt, and James Larus. Deconstructing process isolation. In *Proceedings of the 2006 workshop on Memory system performance and correctness*, MSPC '06, pages 1–10, New York, NY, USA, 2006. ACM.

[3] Ronald T. Azuma. A survey of augmented reality. *Presence: Teleoperators and Virtual Environments*, 6(4):355–385, August 1997.

[4] Ronald T. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre. Recent advances in augmented reality. *Computer Graphics and Applications*, 21(6):34–47, 2001.

[5] D. Chu, A. Kansal, J. Liu, and F. Zhao. Mobile apps: It's time to move up to condOS. May 2011. http://research.microsoft.com/apps/pubs/default.aspx?id=147238.

[6] Loris D'Antoni, Margus Veanes, Benjamin Livshits, and David Molnar. FAST: A transducer-based language for tree manipulation, 2012. MSR Technical Report 2012-123 http://research.microsoft.com/apps/pubs/default.aspx?id=179252.

[7] Micah Dowty and Jeremy Sugerman. GPU virtualization on VMware's hosted I/O architecture. In *First Workshop on I/O Virtualization*, 2008. http://static.usenix.org/event/wiov08/tech/full_papers/dowty/dowty_html/.

[8] Cynthia Dwork. The differential privacy frontier. In *6th Theory of Cryptography Conference (TCC)*, 2009.

[9] Dawson R. Engler and M. Frans Kaashoek. Exterminate all operating system abstractions. In *Workshop on Hot Topics in Operating Systems - HotOS*, 1995.

[10] Adrienne Porter Felt, Serge Egelman, Matthew Finifter, Devdatta Akhawe, and David Wagner. How to ask for permission. In *USENIX Workshop on Hot Topics in Security*, 2012.

[11] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. Android permissions: User attention, comprehension, and behavior. In *Symposium on Usable Privacy and Security (SOUPS)*, 2012.

[12] Robert Hansen. Clickjacking, 2008. http://ha.ckers.org/blog/20080915/clickjacking/.

[13] Emma Hutchings in psfk. Augmented reality lets shoppers see how new furniture would look at home, 2012. http://www.psfk.com/2012/05/augmented-reality-furniture-app.html.

[14] Eliza Gauger in WIRED Magazine. Sonicstyle grey goo cripples second life, 2006. http://www.wired.com/gamelife/2006/11/sonicstyle_grey/.

[15] M. Frans Kaashoek, Dawson R. Engler, Gregory R. Ganger, Hector M. Briceno, Russell Hung, Davied Mazieres, Thomas Pinckney, Robert Grimm, John Jannotti, and Kenneth Mackenzie. Application performance and flexibility on exokernel systems. In *SOSP*, 1997. http://stanford.edu/~engler/sosp-97.ps.

[16] Layar. Layar catalogue, 2013. http://www.layar.com/layers.

[17] LSL Labs Wiki. Grey goo fence, 2012. http://lslwiki.net/lslwiki/wakka.php?wakka=GreyGooFence.

[18] Kevin Poulsen WIRED Magazine. Hackers assault epilepsy patients via computer, 2008. http://www.wired.com/politics/security/news/2008/03/epilepsy.

[19] Edmund B. Nightingale, Orion Hodson, Ross McIlroy, Chris Hawblitzel, and Galen Hunt. Helios: Heterogeneous multiprocessing with satellite kernels. In *SOSP*, 2009. http://research.microsoft.com/apps/pubs/default.aspx?id=81154.

[20] M.Z. Poh, D.J. MacDuff, and R.W. Picard. Advancements in non-contact, multiparameter physiological measurements using a webcam. *IEEE Trans Biomed Engineering*, 58(1):7–11, 2011.

[21] Project Glass. https://plus.google.com/+projectglass/posts.

[22] Qualcomm. Augmented Reality SDK, 2011. http://www.qualcomm.com/products_services/augmented_reality.html.

[23] Franziska Roesner, Tadayoshi Kohno, Alexander Moshchuk, Bryan Parno, Helen J. Wang, and Crispin Cowan. User-driven access control: Rethinking permission granting in modern operating systems. In *IEEE Symposium on Security and Privacy*, 2011.

[24] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from a single depth image. In *Computer Vision and Pattern Recognition*, June 2011.

[25] Paul Viola and Michael Jones. Robust Real-time Object Detection. In *International Journal of Computer Vision*, 2001.