Roadblocks for building mobile AR apps

Jens de Smit, Layar (jens@layar.com)
Ronald van der Lingen, Layar (ronald@layar.com)

Abstract

At Layar we have been developing our reality browser since 2009. Our goal is and always has been to develop a mobile application that makes augmented reality usable and attractive to a large audience. One of the most difficult problems we have faced and continue facing in building a usable, attractive application is is not the challenge of rendering digital content onto the camera view, but dealing with differences between all the mobile devices that users own and we wish to support. Not only the obvious differences between different operating systems are costing us a lot of time and trouble, but also (perhaps even more so) the minute differences between what should have been identical implementations of the same standard or the lack of a standard solution for a common functionality, present serious difficulties on our journeys from great concept to great real world applications. In this position paper we will attempt to point out as clearly as possible the development problems that we are facing on a daily basis, in an attempt to communicate and cooperate with the mobile device industry to get to better mobile AR applications on better mobile platforms.

Introduction

For a full Augmented Reality implementation on mobile devices, many different hardware components and APIs are required. We will go into the details of some issues we encountered while developing Layar for each of these areas

- Camera
- Sensors
- Rendering and compositing
- Device and owner information

Camera

In early versions of Layar, we used the camera image as a backdrop in our augmented reality view. The positioning of the actual AR content was based on the GPS, accelerometer and compass. In order to accurately position this content on top of the camera image, we need to simulate the camera properties in the OpenGL projection.

One of the key challenges in supporting a wide variety of devices is that each device has a different camera and thus can have different optical characteristics (e.g. focal length).
Unfortunately, most mobile platforms do not offer reliable APIs to query these camera parameters. For platforms with a limited number of devices (IOS), this can be overcome by hard coding the parameters. On Android however, the amount of devices is vast, and calibrating all these devices, and maintaining a proper database of camera parameters is a daunting task. A standardised way of reporting the optical characteristics of the camera would greatly benefit the accuracy of AR applications.

For AR applications to be immersive, it is important to maximise screen real estate usage. Ideally, we would show a full screen camera image with augmented content overlayed. On most devices however, the camera preview only supports a select set of resolutions. In order to get a full screen image, we would need at least a preview resolution that matches the aspect ratio of the screen, but this is not always available. That leaves us with the option to show black borders or to stretch the camera image.

With the introduction of Layar Vision, this problem has become a bit more apparent. As the computer vision algorithms are quite heavy for the lower to mid range devices, we want to use a lower resolution camera image for processing. On most Android devices however, there are no lower resolution preview formats available with the same aspect ratio as the screen.

The computer vision algorithms in Layar Vision operate on the live camera feed and attempt to track known images. The quality of tracking is highly dependent on the quality of the camera and on how the camera handles motion and different lighting conditions. To optimize the algorithms, access to the exposure settings of the camera would be helpful. Unfortunately, this is not commonly available on today’s mobile devices.

Sensors
For geo-based augmented reality (and to a lesser extent vision based AR), good reliable motion sensors are essential. Information from the GPS, accelerometer, gyroscope and compass are all combined in order to get an accurate view of the location of the user and the direction they are looking in with their device.

The orientation of the device as calculated from the sensors is used to overlay the augmented content on top of the camera image. However, while testing on different devices, it is noticeable that there are slight differences in the end result. For example, we have noticed differences in horizon height and rotational errors while holding devices in portrait or landscape mode.

Part of this problem is resulting from the fact that the orientation of the sensors in the hardware itself is not the same for all devices. Also the location of the sensors relative to the center of the screen and the camera is not known so it is not possible to compensate in software.

On Android, we have also encountered some issues with sensor drivers. The Android API specification does not specify any hard sensor update frequency requirements. There are only rough hints you can give to the sensor API regarding the required updates frequency.
(DELAY_FASTEST, DELAY_GAME, DELAY_NORMAL, DELAY_UI). This does not give you any insight or control over the actual sensor reading frequency. Different devices will have different sensor update frequencies, so our sensor fusion algorithms needed to deal with the different scenarios.

We also occasionally encounter bigger problems with the sensor output. When first testing Layar on the LG Optimus 2X, we noticed that the AR view kept rotating at high speed. Further investigation proved that the gyroscope sensor was outputting its values as degrees per second, instead of as radians per second as documented in the Android SDK.

Rendering and compositing
Any augmented reality application will have to combine three important elements into a single view. At the lowest level, there is the camera view. An AR content overlay has to be drawn (e.g. in OpenGL) on top of that. Finally, a set of UI components has to be added for controlling the application. When porting Layar to the different platforms, we have run into several issues for properly compositing these three view elements.

The approach we took in the early versions of Layar for Android and iPhone was to stack the different views. We used the platform native camera preview surface as the base of the view and added both a transparent OpenGL surface and a view based on platform native UI components.

While we mostly got this setup stable, we did run into some issues over time on the Android platform. One of the biggest problems was lack of support of the camera preview in portrait mode. This used to work on certain devices prior to Android 2.0, but since the introduction of the Motorola Droid / Milestone, we noticed that this would not work anymore. This required us to re-implement the AR view in landscape mode and implement our own rotated UI components for displaying the overlays in portrait mode.

While we still have this solution in our current Android application, some improved camera APIs in Android 2.2 made these hacks unnecessary, but until we are ready to drop Android 2.1 and lower, we cannot fully transition.

On some of the other platforms that we attempted to port Layar to, it was not possible to overlay a transparent OpenGL view on top of the camera view. For these platforms it was necessary to write the camera image into a texture to be rendered in the OpenGL view itself.

While this is a clean approach and probably the direction that future implementation of Layar will go into, some issues remain. The camera preview image in general is provided in YUV color space, while OpenGL expects an RGB color space. This means that each camera frame will have to be converted, which has a performance impact. On high resolution devices, it is not acceptable to do this on the CPU. By implementing the color conversion in a fragment shader in OpenGL ES 2.0 and doing it in parallel on the CPU, we got acceptable performance.
Since most recent devices will be supporting OpenGL ES 2.0, this becomes a viable approach, but ideally the conversion step would not be necessary. On iOS, the problem can now be avoided as the camera preview can be requested as RGB. On Android 3.0, a different solution is present which allows the camera preview to be directly rendered into a texture as well. It will however take a while before this version of the OS is widespread on devices in the consumers hands.

**Device and owner information**

Apart from the whole AR experience, Layar also sports a whole search and discovery framework to help end users find AR content. Because Layar started as a geo-only AR experience, a lot of that content has a very strong geographical component and, by extension, a lot of layers have a specific target language and a specific target country. To help users find the most relevant content, Layar’s search mechanisms take into account a user’s country of residence, language of choice and the country where the user currently is.

For this information, the client relies on the user’s settings as reported by the mobile OS, summarized by language and country codes. The most common specifications for these codes are ISO 639-1 for languages and ISO 3166-1 for countries. These specifications are available through libraries on many different platforms for many different programming languages, to enable consistent communication between these platforms.

We have found, however, that not all country codes we read from devices are “officially assigned” according to ISO 3166-1. One class of examples consists of those that are “indeterminately reserved”, such as “UK” and “JA”. “Indeterminately reserved” means that the ISO 3166 maintenance agency (ISO 3166/MA) has recognized these codes as being used by other (usually older) specifications, but are not to be officially used within ISO 3166-1 compliant products. Apparently, this has not stopped some developers from using the incorrect codes. For “UK” it is pretty easy to guess that the developer meant the ISO 3166-1 code “GB”, but for “JA” we are left to guess: the developer could have meant Jamaica as per the reservation in ISO 3166-1, but it could also be the deprecated NATO designation or perhaps the ISO 639-1 language code for Japan.

Of course, we can write substitutions for the problems that we find, but we can only do so after they have occurred, meaning that we already paid the price of a dissatisfied user. Plus, in cases like the “JA” code, we are forced to pick either Japan or Jamaica, while, according to our conjecture, there may be phones in both Japan and Jamaica that identify themselves as being in “JA”, in which case we cannot be correct for all cases.

Things get even more frivolous than using a documented but unassigned variant code: we have also seen country codes containing diacritics, such as the code “İT” (capital I with a grave accent), whose intended meaning we can only guess... the only thing we know is that the accompanying language code was for Turkish. The sad result is that we cannot properly serve...
the users that have devices that send these codes back.

Conclusion
This paper lists a few examples of typical problems we have encountered, and still encounter, while developing a product-quality augmented reality application. In some cases, available standards are not used where they could be; in other situations, there are no standards where we could really use them; and now and again, standards are simply implemented incorrectly. The resulting problems we can sometimes work around, but not always without cost. Other problems we cannot fix on our own and must wait for device and OS manufacturers to tackle before we can move on. With this paper we have tried to list some of the problems that we are still facing and would like to see resolved. Establishing a lasting dialogue to support the expedited solution of future problems would be a very welcome outcome of this workshop.