Leap Motion-based Interaction in Augmented Reality Mobile Applications

Dan-Laurențiu Haranguș
Technical University of Cluj-Napoca, Computer Science Department
Cluj-Napoca, Romania
dan.harangus@student.utcluj.ro

Teodor Ștefănuț
Technical University of Cluj-Napoca, Computer Science Department
Cluj-Napoca, Romania
teodor.stefanut@cs.utcluj.ro

ABSTRACT
This paper describes a new interaction method for Web Augmented Reality applications on a mobile phone. The system uses a Web Server for hosting the application and the demanded plugins and utilities files. The designed solution proposes a representation of the human hands in the space of the application based on the Leap Motion device sensors. A set of hands-based gestures is provided in order to interact with the augmented models in the application. An important aspect regarding gestures is that the Leap Motion API supports both fingers and palm gestures, which allows the definition of a large set of gestures in the interaction engine. This type of user interaction can be used in a large variety of mobile applications, for instance applications for home design. The goal of this application is to prove that mobile applications using augmented reality can be extended beside classic capabilities of a smartphone. To demonstrate this, we chose the domain of the application to be home design.

Author Keywords
Human-computer interaction; Gesture recognition; Augmented Reality; Web Applications

ACM Classification Keywords
(H.5.2) User Interfaces

INTRODUCTION

Augmented Reality (AR) is a variation of Virtual Reality (VR). VR technologies completely immerse the user inside a synthetic environment. In contrast, AR allows the user to see the real word, with augmented objects superimposed upon or composited with the real world [1].

Hardware/software development for mobile devices is also a continuous growing industry, with millions of applications in the main mobile operating systems producer markets. This growth is defined by a lot of factors, but we consider the main factor the user's need for portability. A mobile application should summarize the main features of a web or desktop application, packed in a friendly and attractive user interface.

For a while, the mobile world started to adopt both AR and VR. Developers started to build powerful native applications in various areas, such as design, measurement utilities and, of course, gaming. There are many platforms that facilitate building AR applications for each major OS producer, such as ARCore for Android and ARKit for iOS. Those platforms are a set of APIs that enables your phone to analyze its environment, interact with information and understand the world. Some of the APIs are cross platform, for a shared experience. They basically provide algorithms for all the major aspects that compose an AR application: virtual object modelling, surface detection, collision detection, lights and shadows. The main disadvantage of using these platforms is their demand for powerful hardware, so in our case a high-end mobile phone with AR support. Also, neither of these platforms support a client-server implementation, required if you want to integrate a third-party device.

In this work, we propose a solution that represents a compromise between performance on one hand, and availability and extensibility regarding the integration of other devices and APIs, on another hand.

We are using an API that manages to bring AR capabilities directly into a mobile device browser. The mobile browser and phone should also have AR capabilities. The major advantage of this solution is the objective of this work – implementing the new interaction model.

This implementation supports the demanded client-server architecture required for integrating the Leap Motion device. In this case, the entire solution is hosted on a web server, with the Leap Motion device connected to a workstation. The single responsibility of the mobile device remains to connect with the server via a browser with AR support.

The new interaction model is represented by the human hands projections generated by the Leap Motion device. Those projections have the form of the skeleton of the user hands and can be visualized along with the augmented scene. Alongside with those hands, a set of gestures is defined in order to replace the on screen tap based interaction, so well-known for the majority of mobile applications.

In the next section it is presented the related work. After that, we will present a general overview of the proposed system. Next, it is provided the web AR application main implementation aspects. It is followed by section that describes the new interaction model based on gestures. Finally, we will draw the conclusions and propose future improvements.

RELATED WORK

Because of the lack of direct usages of Leap Motion in mobile applications, we split this section based on the two main aspects of the presented subject: mobile interaction related work and AR related work.

The mobile phones are in a continuous development regarding the hardware used and consequently the user interaction suffered some major changes over the last years. The interaction changed from using joystick, directional keys or scroll-bars to the touch sensitive screens.

But there are also others approaches of changing this nowadays traditional touch-based interaction.

In [2], the authors propose an interaction engine based on the device camera sensor. Continuous tracking of the incoming video is used to estimate direction and magnitude. The direction estimates are used for scrolling events, while magnitude can lead to the zoom level or can be used for shake detection. The problem we see regarding this approach is that a continuous tracking of the incoming video can cause significant energy draining for the device.

The approach proposed in [3] is a bit closer to what we propose with this work – moving the interaction engine from the phone hardware to the user. In the quoted article, the authors discuss an eye-gaze tracking technology for mobile phone. Gaze gestures are a new concept, and the authors confirm that based on their studies, this concept is attractive for the users. The problems of this solution are related to the intensity of the outdoor light and calibration.

Mobile AR applications are facing a growth based on the computational capabilities of the latest generation of smartphones, as it is mentioned in [4]. Most of them are displaying to the users either labels for the real object in the scene or are adding virtual objects.

AR applications using only Web are a relatively new concept and the API is still-in-development, so we can mention only the examples offered by [5].

There is a large amount of mobile native AR applications available in markets on different topics. As example, we would like to mention Pokemon Go for gaming and IKEA Place for home design. Both applications provide a better frame rate and more optimal and precise algorithms for surface detection or placing virtual objects, but they lack in support for integrating other APIs or devices for extra features.

SYSTEM OVERVIEW

Figure 1 presents the communication architecture and the flow of data of the proposed system.

For implementation, we used the Leap Motion device, a Samsung Galaxy S8+ smartphone, Google Chrome Canary version 70-72 for the mobile browser with AR support, Google Chrome Web Server for the PC server and Leap Motion SDK v3.2, latest distribution with support for JavaScript programming. We also added in the server the Leap JavaScript plugins, which provides skeleton hands display, support for on-screen position of the hands and gesture recognition.

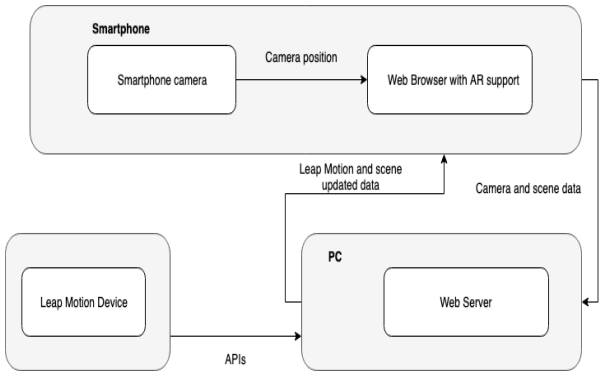

Figure 1. Communication architecture of the system

We obtained two different values for the data flow in the app: one related to the phone browser capabilities of receiving camera data and one related to the operating frequency of the Leap Motion device.

As mentioned above, the phone browser can receive the images from the camera with a rate of approximately 30 frames second. On the other hand, the Leap Motion device works with a rate of approximately 120 frames per second. The frame difference doesn't represent a problem for the application, as the mobile browser will only lose some of the frames sent by the Leap device.

In figure 2 it can be seen the coordinate system of the Leap Motion device. As the Leap device works on 3 axes (X, Y and Z) we need to choose two of them for the 2D on-screen manipulation of virtual objects. We choose X for the left and right movement and Y for the vertical movement of the hand in the space of the device screen.

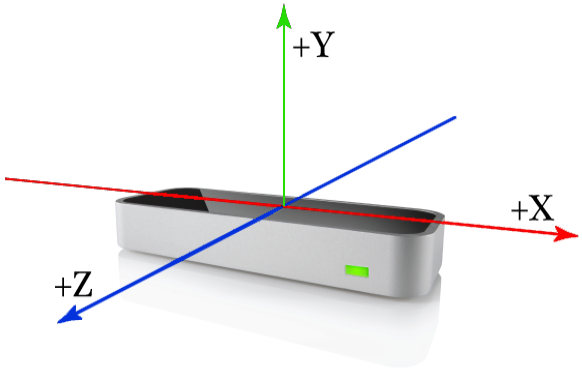

Figure 2. Leap Motion Device coordinate system [6]

Figure 3 briefly presents the set of gestures associated with virtual objects manipulation. This topic will be detailed in the section that describes the interaction model.

Gestures	Object interaction
Index finger pointing	Place object
Pinch	Pick virtual object
Index finger slide	Navigate through object gallery

Figure 3. System set of gestures

AR APPLICATION IMPLEMENTATION

The AR application is implemented using the still-in-development WebXR Device API, the successor of WebVR API, developed in Google Chrome Canary. At this moment, there is a limited range of devices that support this API. It requires an Android device running minimum Android 8.0.0 (Oreo), with ARCore installed and a distribution of Google Chrome Canary between 70 and 72. The bright aspect is that all the browsers with WebVR implementation have committed to support WebXR in the future.

The only feature supported now by the browser is the "hit test" feature. This allows you to cast a ray out from the device and return any collision with the real world, allowing the user to use that information to overlay virtual objects. For the purpose of this system, this feature is all we need. Once we have a virtual object in the screen space we can easily manipulate that object based on his 3D position in the scene space. The actual position of the object is determined by the update of the camera position at each frame. After each update, we can compute the screen position of the virtual object with a simple formula:

```
canvas = renderer.domElement;
if (obj3D.z > 0) {
objScreenPosition = null;
} else {
objScreenPosition.x = Math.round((obj3D.x + 1) *
canvas.width / 2);
objScreenPosition.y = Math.round((obj3D.y + 1) *
canvas.height / 2);
objScreenPosition.z  = 0;
}
```

Canvas object holds the screen size of the device. The check for the negative z parameter of the virtual object 3D position ensures us that the object is currently in front of the camera, otherwise we don't need the transformation. Based on this simple transformation, we can match the object screen position with the Leap hands screen coordinates and alter the object state or position in the real world or even remove that object from the scene.

In case of multiple objects, we designed a menu that always shows currently selected object. To maintain the interaction fully implemented using Leap Motion gestures, we associated the previous mentioned "index slide gesture" to browse the objects in the menu.

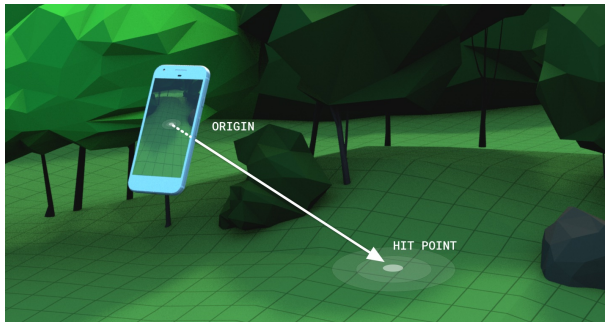

Figure 4. WebXR "hit point" [7]

LEAP MOTION INTERACTION

The Leap Motion tracks hands and fingers. The device operates in the proximity of the user with high precision and a rate of about 120 frames per second. The controller uses optical sensors and infrared light. There are 3 sensors directed along the y-axis when the controller is in standing operating position and have a field of view of about 150 degrees. The range of the device is between 25 to 600 millimeters above the device.

The tracking data model consist of hand and fingers data in its field of view. It provides the updates in a **Frame** of data. The **Frame** is the root data model of the Leap Motion device. The hand model is represented by the **Hand** class. It provides info about the identity, position, the arm attached to the hand, the list of fingers and other characteristics of the detected hand. The *palmNormal* and *direction* vectors define the orientation of the hand. Leap Motion also tracks each finger individually. Fingers are identified by the finger name (i.e. thumb, index, etc). Fingers are represented in the Leap API by the **Finger** class. The *tipPosition* and *direction* vectors provides the fingertip position and the direction the finger is pointing.

Based on this data, we managed to define a set of gestures needed for the interaction with virtual objects. The gestures are inspired by the human real-world sign language and interactions actions.

In order to transform the hand 3D coordinates to the screen 2D space, we used the *screenPosition()* call from the Leap Motion plugin API.

For adding a virtual object in the augmented scene, we defined a pointing gesture using the index finger. The coordinates of this gesture should match the "hit point". Using the hand model, we divided this gesture into 2 main parts: the index finger must be extended and the rest of the hand should be closed. This can be translated in the Leap API by checking the *extended* parameter of the *hand.indexFinger*.

For removing a virtual object, we used the pinch gesture of the *hand.* The coordinates of the pinch should match the screen position of the selected objects. The Leap API *pinchStrength()* call returns a value between 0 and 1, so all

we need to do is to define a minimum value over which we consider a gesture as a pinch.

For switching the selected object in the menu, we defined a sliding gesture using the index finger. For implementing this gesture, we checked if the index finger screen position is inside the menu element and we computed the finger x-axis position from 2 different frames, suggesting a "swipe to right gesture". In order to do this, we proposed a duration of about half a second for this gesture to be completed. In terms of Leap Motion device, a second means 120 frames, so for half a second, we need to check every 60 frames if the **x** coordinate of the *hand.screenPosition()* call is moved from approximately the position of the left border of the menu to the approximate position of the right one and if *hand.indexFinger.extended* returns **true.**

It is important to mention also that the quality of the light do not generally influence the quality of the gesture detection as the Leap Motion device uses a set of infrared cameras.

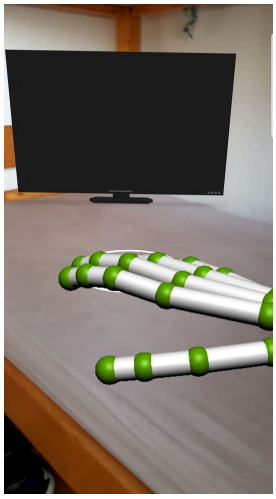

Figure 5. Virtual hand and augmented object

CONCLUSION

We proposed a new interaction model for mobile applications replacing the traditional smartphone screen tapping. We have built a simple AR web application using the WebXR Device API "hit point" feature. We integrated the Leap Motion API in order to use hand gestures for interacting with the virtual objects. The system uses 3 gestures, each mapped to a simple object manipulation function: index finger pointing for placing an object, pinching for removing an object and index finger sliding for menu navigation.

The solution represents a compromise between performance of a native AR development platform and extensibility. By using Web technologies, we managed to offer support for using external devices and third-party APIs.

Through the advantages, it is worth mentioning that the proposed solution brings a constant level of usage difficulty regarding any set of actions, unlike the other interaction models that were analyzed. The main disadvantage is the fact that the system requires a trained user.

FUTURE IMPROVEMENTS

The system does not offer persistence. We are loading the library of objects during the initialization phase of the app and we did not offer the user the possibility of saving the augmented world scene. This aspect can be resolved by using a database for saving the user objects position in the scene.

The user can only interact with the currently selected object from the menu. In case of a complex scene, the user has to scroll repeatedly in the menu in order to switch to the desired object. We need to find an alternative based on the camera field of view and the position of the visible objects.

The interaction between Leap hands and virtual objects is based on matching the hand or tip of the finger and the middle of the virtual object with an addition threshold in each direction. This can be improved by computing each object on screen size and position and check if hands intersect those values.

It has to be mentioned that the system was tested by a relatively small number of users. The application was calibrated after author's preference and the other users were assisted in using it. So, the system may require an interface that allows each user to personalize the default calibration.

REFERENCES

1. Ronald T. Azuma, A Survey of Augmented Reality, IEEE Presence: Volume 6, Number 4, August 1997
2. Haro A., Mori K., Capin T., Wilkinson S., Mobile Camera-Based User Interaction, IEEE Lecture Notes in Computer Science vol. 3766, 2005
3. Heiko Drewes, Alexander De Luca, Albrecht Schmidt, Eye-gaze interaction for mobile phone, IEEE Mobility '07 Proceedings of the 4th International Conference of Mobile Technology applications and Systems and the 1st International Symposium on Computer Human Interaction in Mobile Technology, pages 364-371, 2007
4. Zornitza Yovcheva , Dimitrios Buhalis , Christos Gatzidis, Overview of Smartphone Augmented Reality Applications for Tourism, IEEE e-Review of Tourism Research, Vol.10, No. 2 , 2012
5. Joseph Medley, Augmented Reality for Web, IEEE Chrome Updates, June 2018, https://developers.google.com/web/updates/2018/06/ar-for-the-web
6. Leap Motion device overview, https://developer-archive.leapmotion.com/documentation/javascript/devguide/Leap_Overview.html
7. WebXR Device API hit point diagram, https://codelabs.developers.google.com/codelabs/ar-with-webxr/ - 4