Interaction Toolkit for Programming Interactions with Marker-Based Tangibles in Virtual Reality

João Mesquita, Jorge Cardoso

Interaction toolkit for programming interactions with marker-based tangibles in Virtual Reality

João Mesquita[1][0000-1111-2222-3333] and Jorge C. S. Cardoso[2][0000−0002−0196−2821]

[1] University of Coimbra, DEI joao.diogo106@gmail.com
[2] University of Coimbra, CISUC, DEI jorgecardoso@dei.uc.pt

Abstract. This project focuses on the development of a tangible interaction programing toolkit for Virtual Reality (VR). Marker-based tangible interactions have been explored before, but not development libraries exist for facilitating developers' work. Particularly for accessible web-based VR frameworks such as A-Frame, the complexity of the interactions is limited by programming difficulties, due to the inexistence of high-level programmatic abstractions. As such, in this project the goal is to develop a set of abstractions that facilitate the development of tangible experiences in the A-Frame platform. We have developed an initial version of a component library and have started evaluating it through task-based API usability testing. We report preliminary results from this usability testing.

Keywords: Virtual reality, tangible objects, interaction, programmatic abstraction, visual markers.

1 Introduction

Interaction in Virtual Reality (VR) is often achieved through standard controllers. However, these typically do not provide haptic experiences congruent with the virtual environment and sometimes result in awkward button mappings for manipulating virtual objects.

One alternative is the use of tangible user interfaces which are usually considered to provide a more natural interaction experience by leveraging on our ability to manipulate physical objects and on the direct mappings from physical to virtual objects. Tangibles naturally provide haptic sensations which can increase the realism and thus the immersiveness of the VR experience [1].

The use of tangibles in VR has been accomplished before, using different technological approaches. In this project, we focus on tangible object detection through visual markers, which is a suitable solution for smartphone-based VR. This results in a cheap and accessible tangible system for VR. Previous work by Cardoso and Ribeiro [2] has identified a difficulty in programming tangible interaction given the lack of high-level abstractions for programmers. In this work, we present a marker-based tangible interaction library for the A-Frame web-based VR framework. We also present preliminary results from a usability evaluation of this library.

2 Related work

Previous work by Cardoso and Ribeiro [2] explored the use of tangible objects for interaction in smartphone-based VR, tracking the tangible objects through visual markers. The authors created several prototypes, including a tangible book for exploring cultural heritage and identified a difficulty in programming the VR experiences. Dias et al. [3] have developed a system of controllers with tangible interfaces that allow for hand position tracking. Two prototypes were created, Magic Bracelet and Magic Ring, which consist of visual markers placed on the wrists and fingers of the hand to track the positioning of these body parts and use them for interactions. Lee et al. [4] developed an approach based on occlusion of visual markers in a grid. They implemented fingertip detection, buttons, sliders, two-handed input, over the marker grid. In all these works, the VR/AR experience was programmed from scratch, and no higher-level programming library was created to facilitate the programmer's task of using the implemented interactions.

High-level programming abstraction toolkits such as Phidgets, Reactable, Microsoft Surface, and others [5] exist for several tangible systems, however they are not applicable to marker-based tangible interaction for smartphone-based VR.

3 Tangible Marker Interaction Library

The Tangible Marker Interaction Library is a library of components for the A-Frame web-based VR framework (available at https://github.com/JoaoDiogoMesquita/VR-Tangible-Interaction-Toolkit). It is currently composed of five components: shake detector, button, swipe, and angle detectors (single and double marker). These components follow the general programming approach of A-Frame and trigger high-level events when they detect relevant user actions.

3.1 Shake Detector Component

The shake detector component is intended to detect the shaking motion of a marker. To detect when the marker attached to the object is being shaken, the changes of direction relative to each of the defined axes, the distance traveled and the time between these changes of direction are considered. Programmers can customize the distance and

```
<a-marker … mt-shake-detector="switchInterval: 500;
                               minimumSwitchTimes: 3;
                               minimumDistance: 0.3;
                               eventTargets:#myBox;
                               axis:y;">
</a-marker>
```

Fig. 1. Demonstration of the "shake detector" component.

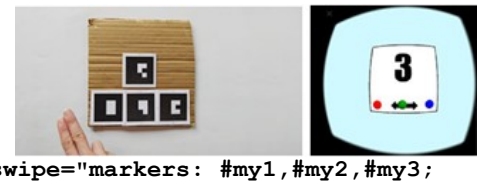

```
<a-marker … id=”ref”>  </a-marker>

<a-marker … mt-button="referenceMarker: #ref;
                       eventTargets: #box;
                       minimumTime: 100;">
</a-marker>
```

Fig. 2. Demonstration of the “button” component.

maximum time interval between changes of direction, as well as the minimum number of times they occur. Programmers can also set which movement axes will be detected: x, y, or z. This component can help in creating interactions where users interact with the virtual environment by shaking objects. For example, a user manipulating a tangible representation of an architectural model of a church tower can shake the tangible and trigger the sound of the tower’s bell (Fig. 1).

3.2 Button Component

The button component follows an approach similar to [5] and allows using a marker as a button that can be “clicked”. It uses two markers: one serves as a reference and must be visible for the interaction to occur; the other serves as the actual button that users can click by touching (occluding it). The reference marker avoids the ambiguity between the occlusion of the button marker occurring due to it moving out of scene vs being explicitly occluded by the user. When the button marker is occluded for a predefined amount of time (configurable by the programmer), a click event is triggered. In a VR experience this interaction can easily be included, acting as a switch or any functionality executed with a button producing the respective behavior. For example, Fig. 2, shows a demo of a tangible object that allows browsing content by pressing on of the lower buttons to move back and forth.




```
<a-marker … mt-swipe="markers: #my1,#my2,#my3;
                      eventTargets: #box1, #box2;
                      sequences:1 2 3,3 2 1;
                      maximumTime: 3000;”>
</a-marker>

<a-marker … id="my1" > </a-marker>
<a-marker … id="my2" ></a-marker>
<a-marker … id="my3" ></a-marker>
```

Fig. 3. Demonstration of the “swipe” component.


```
<a-marker … mt-angle-detector-sm="threshold:45;
                                  eventTargets: #box1, #box2;
                                  axis:y;" >
</a-marker>
```

Fig. 4. Demonstration of the "angle detector – single marker" component.

3.3 Swipe Component

The Swipe component allows detecting swipe interactions on a sequence of markers. The programmer defines the set of markers to be used, as well as the sequences to detect. If the markers are occluded in a predefined order and within the defined maximum time, an event will be triggered. One of the advantages of using this type of interaction is the possibility to customize the behavior, and each defined sequence can be distinguished in the detection, so it is possible to have several different types of behaviors in the same tangible object (swipe left, right, up, down). Fig. 3 shows an example of the use of the Swipe component where three markers are used for swiping left and right to browse contents.

3.4 Angle Detector Component – Single Marker

The Single Marker Angle Detector component detects the rotation of a marker with respect to one or more defined axes. The programmer can set a threshold (in degrees) and an event will be triggered when the marker rotates by this amount. The event contains information about the axis, direction, threshold, and the entity where it occurred. This component can be used to change continuous variables in virtual space such as brightness or volume. Fig. 4 shows an example of the use of the angle detector for creating a rotational controller.

```
<a-marker … id="my1" mt-angle-detector-dm="
                                  threshold: 10;
                                  movement : 1;
                                  secondMarker: #my2;
                                  eventTargets: #box2, #box1;">
</a-marker>

<a-marker … id="my2" ></a-marker>
```

Fig. 5. Demonstration of the "angle detector – double marker" component.

3.5 Angle Detector Component – Double Marker

The Double Marker Angle Detector component handles the measurement of the angle between two markers. The component allows the programmer to customize the range and type of angle to detect. The first type intends to detect the angle between two markers on different planes, such as markers on two pages of a book. The second type of angle assumes the markers are in the same plane, and the calculated angle is the one between the sides of each marker. Given a customizable threshold, events will be issued to the entity associated with the component and to additional specified entities each time this threshold is reached. This component can be used, for example, in the opening movement of a book, producing an animation depending on the opening angle between the two pages (Fig. 5).

4 Usability Evaluation of the Components' API

We are currently performing API usability tests to evaluate the understandability, abstraction, learnability, and reusability of the provided component library. We are following a task-based API usability evaluation [6]. For each component, we ask programmers to solve two tasks. The first task for each component can be implemented by using only the HTML interface of the component. The second task can only be solved by coding a custom A-Frame component in JavaScript and using the component's JavaScript API.

This first usability test is being performed online and asynchronously: participants are given the set of tasks and the library of components and are asked to solve them at their own pace and send us their answers when finished. For this reason, for each task we provided pre-recorded videos of interaction through marker-based tangibles and pre-configured the base source code for the tasks with the marker detection library to use these videos so that participants could focus on the solution and not on creating and manipulating the marker tangibles. As an example of the tasks, for the Shake Detector component the tasks were:

> Task 1: Use the Shake Detector component to detect the marker/box shaking front/back and turn the element with the sphere into green.
>
> Task 2: Detect the marker/box shaking and: move the sphere 1 meters to the right each time the box is shaken horizontally (left/right); move the sphere 1 meters up each time the box is shaken vertically (up/down).

After solving the tasks, we asked participants to answer a questionnaire with demographic questions (age, country, work area, programming experience, and specific programming experience with A-Frame), and questions adapted from [7] to evaluate our library in terms of understandability, learnability, abstraction, and reusability (Fig. 6).

	Question	-2	-1	0	1	2
Understandability	1. Do you find that the API types map to the domain concepts in the way you expected?	0	0	0	1	4
	2. Do you feel you had to keep track of information not represented by the API to solve the tasks?	1	0	1	1	2
	3. Does the code required to solve the tasks match your expectations?	0	0	1	2	2
Learnability	4. Once you performed the first two tasks, was it easier to perform the remaining tasks?	0	0	0	0	5
	5. Do you feel you had to learn many classes and dependencies to solve the tasks?	0	0	0	3	2
Anstraction	6. Do you find the abstraction level on the interactions programed appropriate to the tasks?	0	0	1	1	3
	7. Did you need to adapt the API (like overriding default behaviors in Javacript code) to meet your needs?	0	0	0	1	4
	8. Do you feel you had to understand the underlying implementation of the components to be able to use the API?	0	1	1	1	2
Reusability	9. Does the amount of code required for each task seem about right, too much, or too little for you?	0	0	0	3	2
	10. How easy was it to evaluate your own progress (intermediate results) while solving the tasks?	0	0	1	3	1
	11. Do you feel you had to choose one way (out of many) to solve a task in the scenario?	0	0	2	0	3
	12. Do you feel you would have to change much in your code to change the interaction behaviour?	0	0	1	1	3

Fig. 6. Results from the API usability questionnaire. The results from the original 5-point Likert scale have been transformed to [-2, 2] and adjusted so that positive values are always better.

4.1 Preliminary results and discussion

Five participants have completed the API usability test. Four stated to have between 5-10 years of programming experience, while one stated having between 1-5 years of experience. Four had previous experience with A-Frame, although only two had previously coded an A-Frame JavaScript component.

Fig. 6 presents the results from the usability questionnaire, with the answers normalized so that higher values are better, regardless of how the question was worded. In general, these preliminary results show a positive evaluation. Some items need further analysis since they do not seem to have a clear general answer (e.g., item 2 and 8, which had at least one negative answer). The learnability dimension seems to have had the best results, although this is perhaps not surprising as the library is not currently very extensive and so a few examples should be enough to get the overall gist for how it works. These answers need to be analyzed together with the source code submitted by the participants and their additional comments. As a complement to this remote usability test, we are also considering performing in-person tests to be able to better observe and understand the difficulties faced by programmers of our library.

5 Conclusion and Future Work

The objectives of this project are to develop a tool that allows developers of VR experiences to incorporate marker-based tangible interaction more easily in their projects. We have developed a library that incorporates five components, so far. These components are inspired by previous work that used marker-based tangibles.

At this moment, the project is in a testing phase, with task-based usability tests being conducted. Although preliminary results are satisfactory, the data needs further analysis for us to be able to determine how to improve the proposed library.

References

[1] K. Hinckley, R. Pausch, J. C. Goble, and N. F. Kassell, "Passive real-world interface props for neurosurgical visualization," Conf. Hum. Factors Comput. Syst. - Proc., pp. 452–458, 1994, doi: 10.1145/191666.191821.

[2] Cardoso, J. C. S., & Ribeiro, J. M. (2021). Tangible VR Book: Exploring the Design Space of Marker-Based Tangible Interfaces for Virtual Reality. Applied Sciences, 11(4), 1367. https://doi.org/10.3390/app11041367

[3] J. M. S. Dias, N. Barata, P. Santos, A. Correia, P. Nande, and R. Bastos, "In your hand computing: Tangible interfaces for mixed reality," in ART 2003 - IEEE International Augmented Reality Toolkit Workshop, 2003, pp. 29–31, doi: 10.1109/ART.2003.1320422.

[4] G. A. Lee, M. Billinghurst, and G. J. Kim, "Occlusion based interaction methods for tangible augmented reality environments," in Proceedings VRCAI 2004 - ACM SIGGRAPH International Conference on Virtual Reality Continuum and its Applications in Industry, 2004, pp. 419–426, doi: 10.1145/1044588.1044680.

[5] C. Moussette, "Tangible interaction toolkits for designers," Scand. Student Interact. Des. Conf., pp. 2–5, 2007, [Online]. Available: https://www.netlearning2002.org/tek/sider07.nsf/(WebFiles)/FE9AD2C7730323FAC12572A900576009/$FILE/SPA_033.pdf.

[6] Rauf, I., Troubitsyna, E., & Porres, I. (2019). A systematic mapping study of API usability evaluation methods. *Computer Science Review*, *33*, 49–68. https://doi.org/10.1016/j.cosrev.2019.05.001

[7] Piccioni, M., Furia, C. A., & Meyer, B. (2013). An Empirical Study of API Usability. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 5–14). IEEE. https://doi.org/10.1109/ESEM.2013.14