# A-Frame experimentation and evaluation for the development of interactive VR: a virtual tour of the Conimbriga Museum

**TECHNICAL REPORT, MAY 2019**

Solange Gomes Santos
solangegs@student.dei.uc.pt

Jorge C. S. Cardoso
jorgecardoso@ieee.org

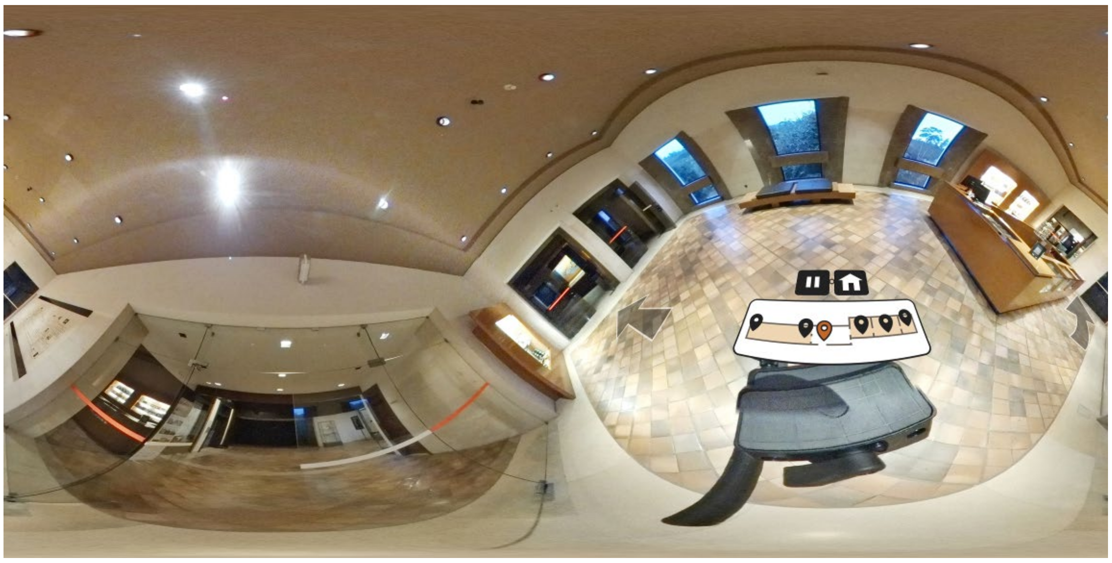
CISUC, Department of Informatics Engineering, University of Coimbra, Portugal

## ABSTRACT

This work provides an analysis of the development effort required to implement a Web-based VR experience using the A-Frame VR framework. We implemented an interactive 360º virtual tour of the Conimbriga Museum, featuring the objects of national interest that can be found in the museum's exhibition. The tour allows navigation between the 360º scenes, providing information panels about the objects, and a floor menu for rapid teleporting between rooms. The implementation of this virtual tour allowed us to identify several issues where Web-based VR development is still lacking in flexibility.

**Keywords**

Virtual reality, web development, A-Frame, virtual tour, 360º image, interactive tour



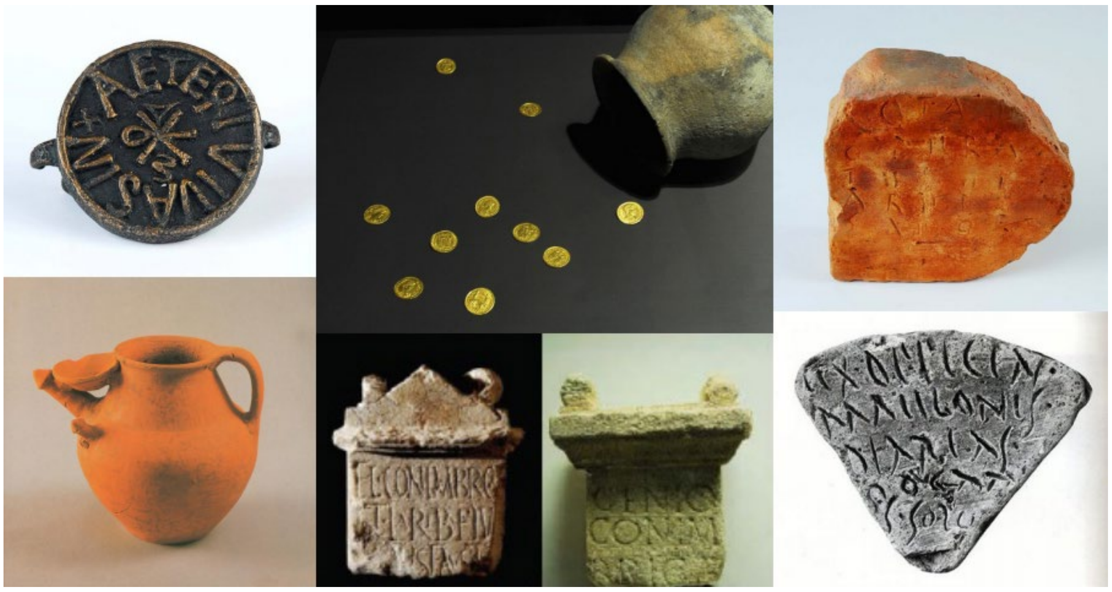


Figure 1: Pieces of national interest from the Conimbriga Monographic Museum. Top left: liturgical seal; top middle: solidi's treasure; top right: magic square; bottom left: ritual vase; bottom middle: votive altar; bottom right: brick with inscription.

## INTRODUCTION

Virtual Reality (VR) is a term that is increasingly known and present in our daily lives. Science fiction works have speculated amazing scenarios, some of which may have conveyed the feeling that VR is not an attainable technology for the regular person. However, over the past few years this technology has been increasingly refined, and it has quickly become possible and easy to enjoy VR in the comfort of our own house using accessible equipment.

As the enjoyment of VR experiences has become something attainable, so should the tools to create these experiences follow the same logic and become increasingly simple and accessible for anyone to use. There are already several tools that allow the creation of VR experiences such as Unity 3D (Unity Technologies, 2019), Unreal Engine (Epic Games, 2019) and React 360 (Facebook Inc., 2019). These are, however, primarily targeted at knowledgeable programmers. In this project, we analyze and evaluate a web-based VR development framework named A-Frame (A-Frame, 2019).

A-Frame is an open-source web framework for building VR experiences based on HyperText Markup Language (HTML) and JavaScript, originally developed within the Mozilla VR team. It was developed to be an easy and powerful tool for creating VR content on the Web. HTML is an easy to read and understand markup language making A-Frame an accessible framework to non-programmers, particularly Web developers. Additionally, A-Frame is multi-platform, supporting VR headsets such as HTC Vive, Rift, Windows Mixed Reality, Daydream, GearVR and Google Cardboard.

The purpose of this work was to evaluate the capabilities of A-Frame and the required development effort to create a 360º interactive virtual tour by a content creator with a Web development background. We consider the use of 360º images an easier and more straightforward way to build a VR environment, since it does not require 3D modelling capabilities. With the environment created, one can then add interactions with elements of the images or create new virtual elements that fit into the environment. For this reason, we are particularly interested in evaluating A-Frame in the development of VR environments with the use of 360º image and incorporation of interactive elements, as well as the adaptation of these environments to different VR platforms.

To serve as a case study, we developed a virtual tour for the Conimbriga Monographic Museum. Located in Condeixa-a-Nova, Portugal, the museum has a collection composed exclusively of archaeological material found in the ruins of the Roman city of Conimbriga. The current exhibition displays everyday objects organised by theme. Six of these objects have been classified as of national interest (Figure 1). The virtual tour highlights these objects of interest and allows users to interact with them to see additional details.

The following process was used in this work.

- Learning A-Frame through an existing online course[1] and through the official A-Frame's documentation[2].
- Listing necessary virtual tour elements by analysing existing virtual tours based on 360° photos to uncover the main elements that would be necessary to implement.
- Implementing each virtual tour element separately. In some cases documented below, variations of a given element were implemented.
- Integrating all the implemented elements in a virtual tour.

This work was done in the scope of a research grant by CISUC/DEI. The resulting GIT repository can be found at https://git.dei.uc.pt/jorgecardoso/conimbriga-360.

## RELATED WORK

### X3DOM

X3DOM (Behr, Eschler, Jung, & Zöllner, 2009) merges the declarative language of Extensible 3D Graphics (X3D) with the Document Object Model (DOM), providing a way to create "declarative 3D scenes in Web pages" (Fraunhofer-Gesellschaft, 2019) without users needing to install any browser plugin (it is based on HTML5 and WebGL). In its goals, X3DOM is very similar to A-Frame and could be considered more mature, since it is being developed since 2009. In this work, we opted to work with A-Frame simply because it provides out-of-the-box support for WebVR enabling it to interface with VR devices such as HTC Vive, Oculus Rift, etc., without having to write any specific code for that.

### React 360

React 360 (Facebook Inc., 2019) is a framework for the creation of 3D and VR user interfaces. It is built on top of the React library – a JavaScript library for building user interfaces. As a JavaScript library, React provides a very different way to building webpages than standard HTML. Even though React has a large developer base, it is still a specialised skill that not all Web developers possess.

[1] https://aframe-uc.glitch.me/index.html

[2] https://aframe.io/docs/0.9.0/introduction/

## THE A-FRAME FRAMEWORK

An A-Frame project is simply an HTML file, just like a typical web page. The A-Frame framework can be used without requiring any actual installation, as we can include A-Frame's JavaScript file in the HTML document through a <script> tag pointing to the CDN (Content Delivery Network) distribution:

```
<script src="https://aframe.io/releases/0.8.0/aframe.min.js"></script>
```

To create a VR environment, we insert the <a-scene> tag, which will contain all the necessary components of the scene.

As a first contact with A-Frame, we tried to create basic elements, starting with 3D primitives such as planes, cubes, spheres, cylinders, background, and many others. Creating these elements has proved to be a simple task, since it is only necessary to add a tag and specify primitive properties such as position, size, and color. For example, to create a sphere, we simply add a <a-sphere> tag as follows:

```
<a-sphere position="3 3 -10" radius="0.7" color="red"></a-sphere>
```

The addition of other elements such as lights, text, portals to other locations, proved to be equally simple, as they follow the same logic.

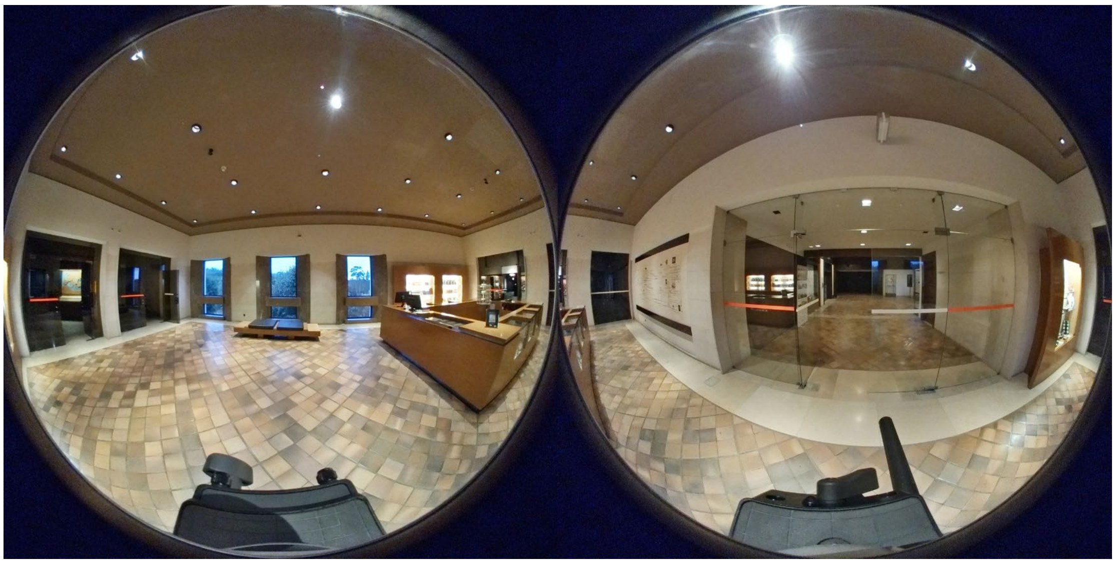
We also attempted to create basic interactions with the elements that can be activated by events such as "click", "mouseenter", "mouseleave", and allow users to change properties of the element with which they are interacting. We also experimented with making animations that can be created by defining the duration of the animation, the delay, and the property to be changed, for instance, changing the color of an element from red to blue. At the moment this project was developed, animations required the use of the "aframe-animation-component" (https://www.npmjs.com/package/aframe-animation-component).

Creating a scene and adding elements to it with A-Frame, as well as basic interactions and animations, is a fairly simple process, being very similar to building a typical web page in HTML. However, it is necessary to test the construction of virtual environments with more complex interactions to understand how the development effort increases as the environment gains complexity.

Thus, with the basic knowledge for using A-Frame acquired, we began to make implementation tests of features that could be useful in the context of a virtual tour, namely the creation of pop-ups to appear through interaction with some objects, the creation of a menu with a map to go to various points of interest and the ability to control the sound of the virtual experience, navigation between 360º images through arrows or portals, among others.

In performing these implementation tests, we realized that it isn't possible to create complex elements using only the HTML elements provided by A-Frame. In fact, in order to implement some features, it was necessary to register new A-Frame components through JavaScript, so some knowledge of JavaScript and understanding of how A-Frame works and its elements interact is needed.

After testing these features, we started to develop the museum virtual tour prototype, using much of the implementation that we previously tested in its development. We explain below in more detail how each feature was implemented and the effort it required.


(a) spherical format

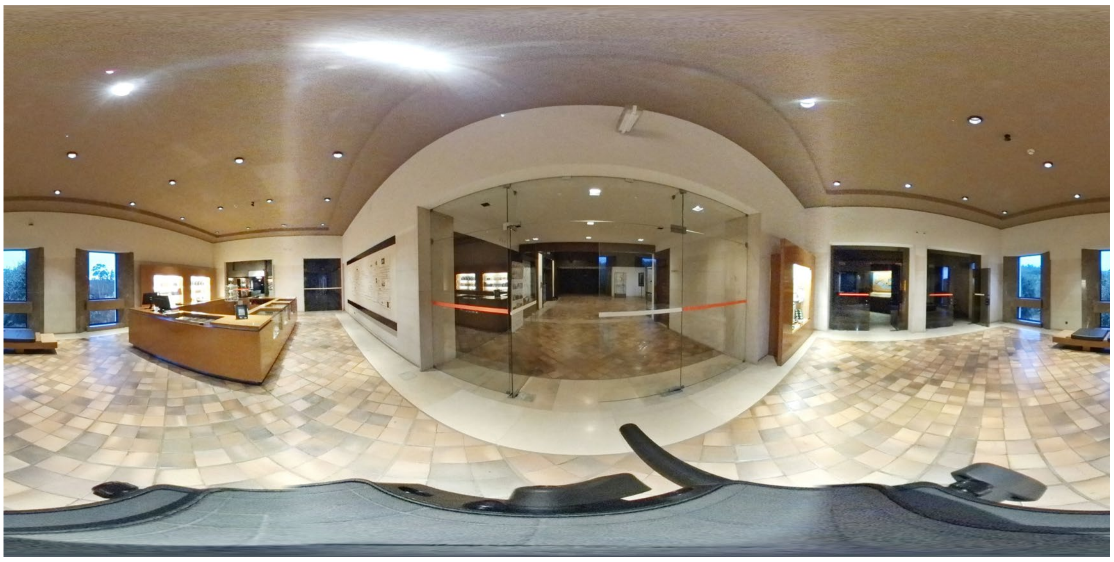

(b) flat panoramic format

Figure 2: 360º image format taken by the camera (a) and after conversion (b).

## DEVELOPING A 360º VIRTUAL TOUR

As we intend to experiment and evaluate A-Frame in the development of interactive VR in a 360º image environment, we developed a virtual tour for the Conimbriga Monographic Museum that followed the subsequent steps:

1. 360º photographs: acquisition of 360º photographs at various points inside the museum rooms and construction of the 360º environments.

2. Navigation between scenes: implementation of navigation between the 360º images through arrows and circles.

3. Navigation menu: creation of a menu with the museum map having its main points marked.

4. Highlighting of museum objects and interaction: highlighting the national interest objects and interaction with these to trigger informational pop-ups.

5. Portals: creation of portals leading to provenance ruins of the pieces.

Each of these steps is described in detail below.

## 360º photographs

The Conimbriga Monographic Museum is located next to the ruins of the Roman city and is composed by a reception room and four rooms with exhibition of archaeological material found in the Roman city.

To acquire 360º photographs of the museum we used a Samsung Gear 360 camera placed on a tripod, aligned to the average height of a person's eyes – around 165 cm. We went to the museum on a quiet period (with few visitors so we could take photographs of the rooms without any people) took several photographs in main points of the rooms, namely in front of each showcase and near entrances. Having an image of the museum floor plan at hand, we marked the various points where we were taking photographs in order to have a better understanding of the amount of points we had already registered and the coherence they had between each other and between each room.

Although we took more pictures, we selected 26 for the final virtual tour. The images recorded by the camera have a spherical format by default and need to be converted to flat (equirectangular) panoramic images (see Figure 2).

After converting the 360º images to the correct format, we can use them in our A-Frame project. The process for this is quite simple: it is necessary to include the image in the assets[3] list of the scene and specify the image to be used inside the <a-sky> tag, which represents a large sphere with a color or texture mapped inside, as in the following example:

```
<a-assets>
   <img id="sky" src="sky.png">
</a-assets>
<a-sky src="#sky"></a-sky>
```

Thus, to start building our virtual tour, we created an HTML file for each 360º photograph to be used and repeated this process for each one.

## Navigation between scenes

In the context of a virtual tour with navigation between 360º images, it makes sense to teleport directly from a point to others next to it – similarly to what happens when we navigate the streets in Google Maps. For this, it is important that the possible navigational points are well identified and that it is understandable that such interaction is possible.

In order to teleport from one point to another, that is, from one 360º scene to another, we need to use the <a-link> tag. This tag can be defined as a portal and behaves very similarly to links in a web page. This means that when a portal is clicked, it takes the user to another virtual world, defined in a different file. These elements, by default, are represented as

[3] Using the assets element is not mandatory, but it generally a good practice, particularly if the assets are used more than once in the virtual scene.

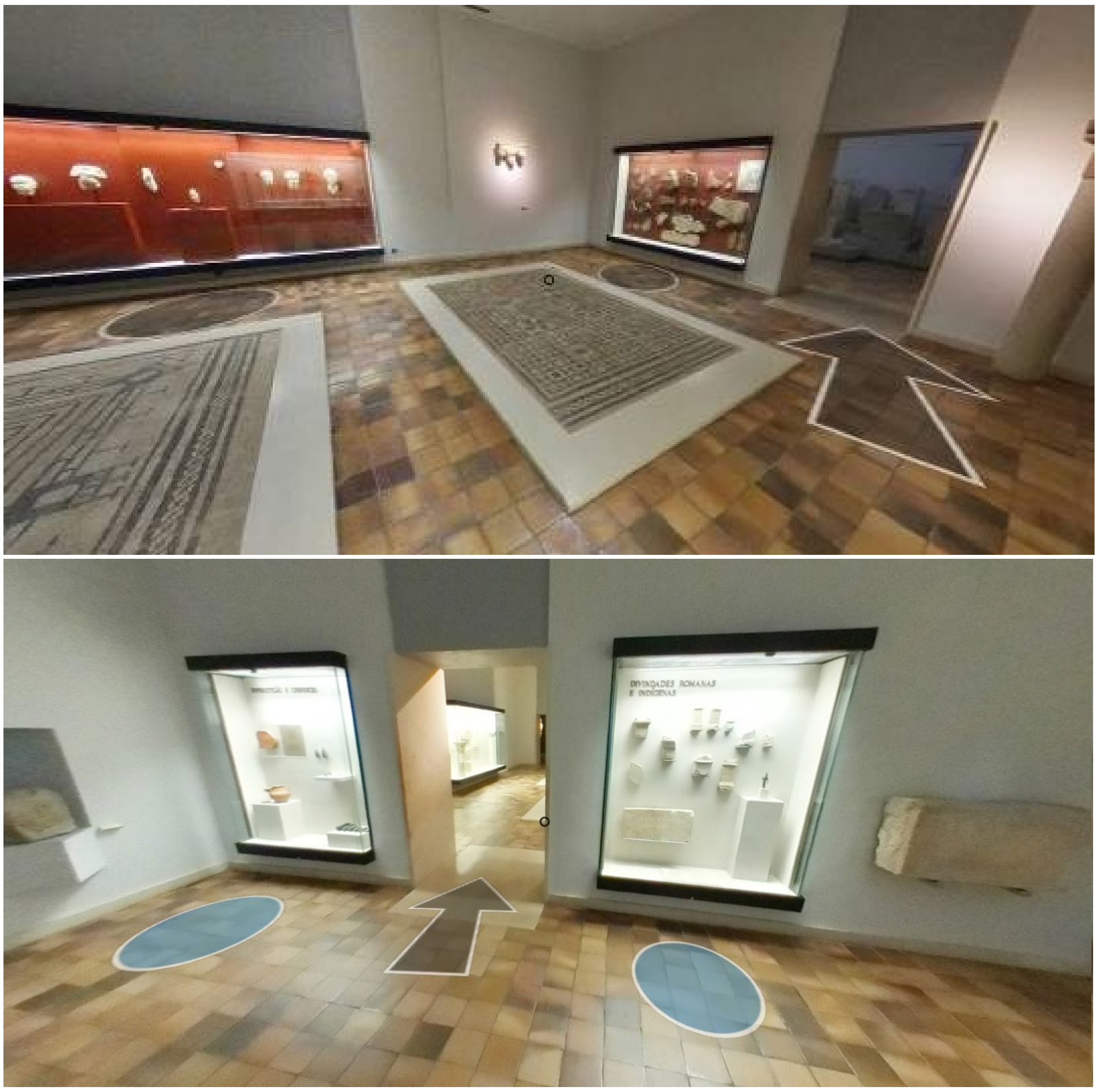


Figure 3: Navigation between 360º worlds through points and arrows.

circles with a title over them and a graphical preview of the world to which they redirect; however, this appearance can be easily disabled.

In the case of our virtual tour, we did not intend to use the default appearance, but instead use other elements as navigation signs, namely arrows and points. For this purpose, we can create the images we want to use and insert them inside the portal as follows:

```
<a-link position="0 0 0" link="visualAspectEnabled:false"  href="another_world.html">
   <a-image src="#arrow" rotation="-90 0 -7" position="1 0 -3" height="1.9" width="1.2">
   </a-image>
</a-link>
```

This allows the portal to take the shape of the image embedded within it, which in this case is the shape of an arrow. Note that the portal should be placed in 3d space, so it is necessary to position these elements in the scene with enough accuracy, as well as there may be a need to adjust its rotation and/or size.

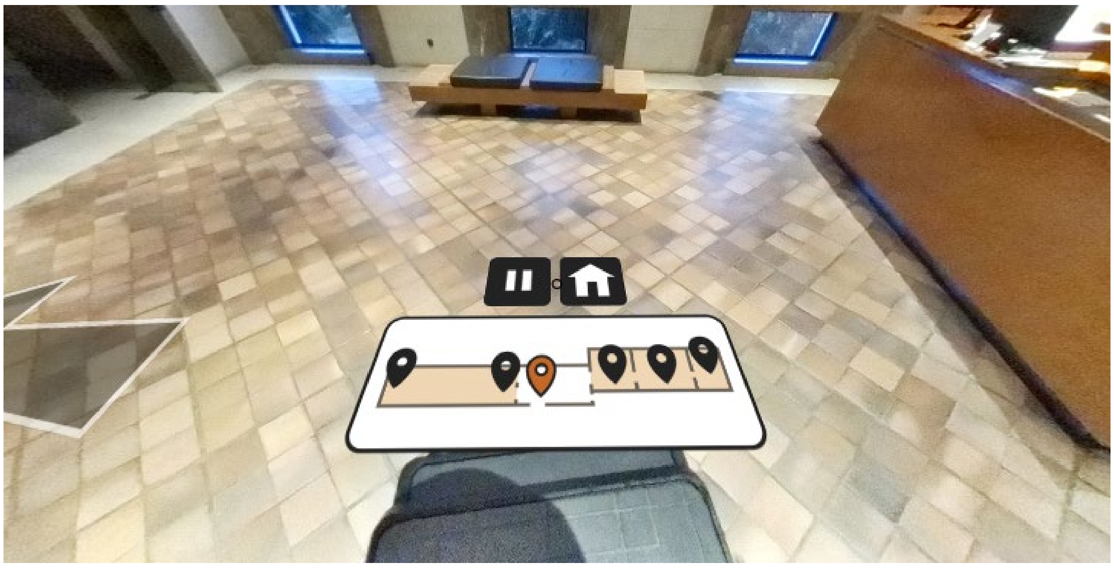


Figure 4: Menu with buttons and map.

During the development of navigation between the 360º images, we made some important interaction decisions that we considered to improve the navigation experience (Figure 3):

1. We use circles to identify floor locations that the user can clearly see when teleporting;

2. We use arrows to identify the direction of movement to locations which the user cannot see (for example, moving from one room to another);

3. We use elements with a different color, in this case blue, to identify locations where any of the objects of national interest is displayed;

4. We apply a rotation to the 360º world to which we are being teleported, depending on the point we were before, in order to create a more natural navigation flow and allow the user to be better positioned in space.

The first and second decisions were made since we have these two possibilities – places that the user can see clearly from the current perspective in the scene, and places that the user cannot see directly. The second situation occurs for example when structures such as walls and doors prevent us from seeing the exact location on the floor where we are going to be teleported. The arrows used in these situations are meant to provide an indication of the virtual movement direction instead of indicating exactly where the user will be teleported.

The third decision came as an extra support to help the user identify the places where objects of national interest are located, as these are scattered throughout the museum and it may not easy to identify them without visual aids. Although we highlight these objects in other ways, as we will discuss later, this solution is another hint on how to get to these objects.

Finally, the fourth decision was taken because when entering a virtual scene, the user is placed in a default perspective, i.e., always facing a given direction. When traversing links, users should enter the new scene and face a direction that is consistent with the direction they were facing in the previous scene. By default, this will not happen, and users may be teleported into a new scene facing a different direction. For example, users could click a link to go through a door into another room and be placed facing the same door they came in through. To solve this, we apply a rotation to the world, depending on which link the user entered the scene through. To apply this rotation, we added the "rotation" parameter in the scene’s URL and indicated the rotation required for each case, as in the following example:

```
<a-link position="0 0 0" link="visualAspectEnabled:false" href="2.html?rotation=-90">
   <a-image src="#arrow" rotation="-90          0          -7" position="1          0          -
3" height="1.9" width="1.2" opacity="0.6" event-set__mouseenter="opacity:          1" event-
set__mouseleave="opacity: 0.6">

   </a-image>
</a-link>
```

When we move to another virtual world, we read the assigned rotation of the URL and apply that rotation to the world through JavaScript. It should be noted that this is a partial solution that only works well for gaze-based interaction where we can be certain that the user is facing a specific direction when clicking the link (because clicking is accomplished by looking at). If users can click the link while facing a different direction our solution will still not fully work. For a full solution, the “rotation” parameter should be established dynamically when the user clicks the link.

## Navigation menu

For virtual tours where there are various rooms to visit, especially if there are many rooms – it is essential to provide a navigation map for quick access to a given room (instead of forcing users to go through all the locations in between. A navigation map also provides users with context about where they currently are in the museum.

In this project, we included a simple map that shows all 5 rooms in the museum – the museum lobby, plus the four exhibition rooms. Users can teleport to a pre-defined location in each room – except for the left room which is larger than the others and so users can teleport to one of two possible locations. The current room is highlighted in the map with an orange icon, allowing users to quickly understand where they are in museum (Figure 4).

The navigation map is part of a larger “floor menu” – a menu that appears when users look down at the floor. This floor menu includes also the ability to enable or disable audio narration and the possibility of teleporting to a home location (the main lobby of the museum).

In the HTML, this menu is created with an <a-entity> element within which all menu images and links are placed. The following code excerpt shows the images and links for the home, play, and pause buttons and the first point on the map.

```
<a-entity id="menu" position="0 0.2 0" follow="target: a-camera">
   <a-image id="map_image" src="#map" rotation="-90 0 0" position="0 0 -0.65"
      height="0.65" width="1.7">
   </a-image>
    <a-entity play-pause>
       <a-image id="play_button" src="#play" rotation="-90 0 0" position="-0.2 0 -1.2"
          height="0.35" width="0.35">
      </a-image>
       <a-image id="pause_button" src="#pause" rotation="-90 0 0" position="-0.2 0 -1.2"
```

```
        height="0.35" width="0.35">
      </a-image>
    </a-entity>
    <a-link position="0 0 0" link="visualAspectEnabled:false" href="1.html">
      <a-image id="home_button" src="#home" rotation="-90 0 0" position="0.2 0 -1.2"
        height="0.35" width="0.35">
      </a-image>
    </a-link>
    <a-link position="0 0 0" link="visualAspectEnabled:false" href="1.html">
      <a-image id="location1" class="location" src="#current_location" rotation="-90 0 0"
        position="-0.06 0.1 -0.6" height="0.2" width="0.14"
        event-set__mouseenter="height: 0.23; width: 0.16"
        event-set__mouseleave="height: 0.2; width: 0.14">
      </a-image>
    </a-link>
    [....]
  </a-entity>
```

Notice how “event listeners” where added so that, when the interaction cursor is over a point on the map, the respective icon grows providing the user with an interactivity cue.

If this floor menu was always visible it would obstruct too much the museum space and would possibly become annoying and distracting to the user. To make the menu appear only when users look down, we implemented a custom A-Frame component:

```
AFRAME.registerComponent('follow', {
  schema: {
    target: {type: 'selector'}
  },

  init: function () {
  },

  tick: function (time, timeDelta) {
    var targetYRotation = this.data.target.object3D.rotation.y;
    var targetXRotation = this.data.target.object3D.rotation.x;

    if(targetXRotation < -0.80){
      if(!is_menu_visible){
        this.el.object3D.rotation.y = targetYRotation

        if(playing){
          pause_button.setAttribute("opacity", "1");
        }
        else{
          play_button.setAttribute("opacity", "1");
        }

        home_button.setAttribute("opacity", "1");
```

```
            map_image.setAttribute("opacity", "1");
            for (var i = 0; i < locations.length; i++) {
                locations[i].setAttribute("opacity", "1");
            }
            is_menu_visible = true;
        }
    } else {
        play_button.setAttribute("opacity", "0");
        pause_button.setAttribute("opacity", "0");
        home_button.setAttribute("opacity", "0");
        map_image.setAttribute("opacity", "0");
        for (var i = 0; i < locations.length; i++) {
            locations[i].setAttribute("opacity", "0");
        }
        is_menu_visible = false;
    }
  }
});
```

The component was named “follow” and was attached to the menu entity by adding an attribute with the same name:

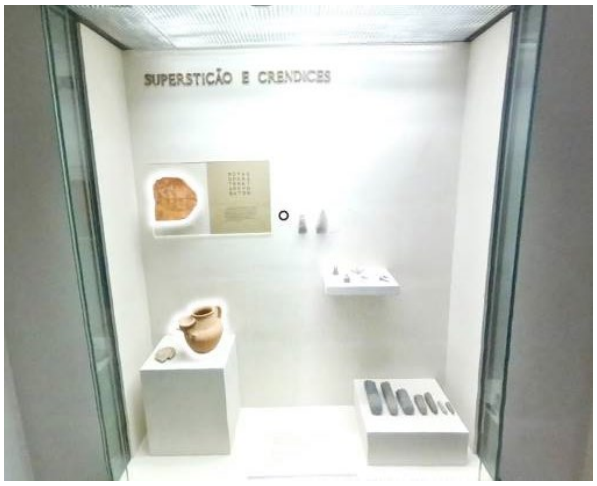
```
<a-entity id="menu" position="0 0.2 0" follow="target: a-camera">
```

The component works by “following” the camera, i.e., by updating itself to always be in the floor in front of the camera. If the user looks down (X rotation less than 45º or 0.80 radians) and the menu is not yet visible, than the menu’s rotation is updated to match that of the camera:

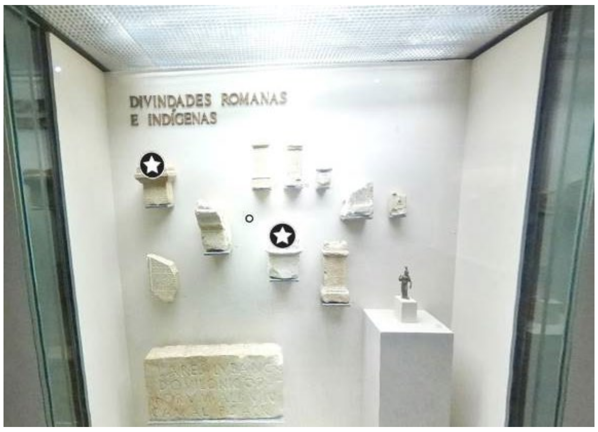
```
var targetYRotation = this.data.target.object3D.rotation.y;
var targetXRotation = this.data.target.object3D.rotation.x;
if (targetXRotation < -0.80){
    if (!is_menu_visible){
        this.el.object3D.rotation.y = targetYRotation
```

Next, we make all menu elements visible:

```
If (playing){
    pause_button.setAttribute("opacity", "1");
} else {
    play_button.setAttribute("opacity", "1");
}
home_button.setAttribute("opacity", "1");
map_image.setAttribute("opacity", "1");
for (var i = 0; i < locations.length; i++) {
    locations[i].setAttribute("opacity", "1");
}
is_menu_visible = true;
```



(a) white highlight



(b) stamp

Figure 5: Evidencing objects through highlighting (a) and through a stamp (b).

## Highlighting of museum objects and interaction

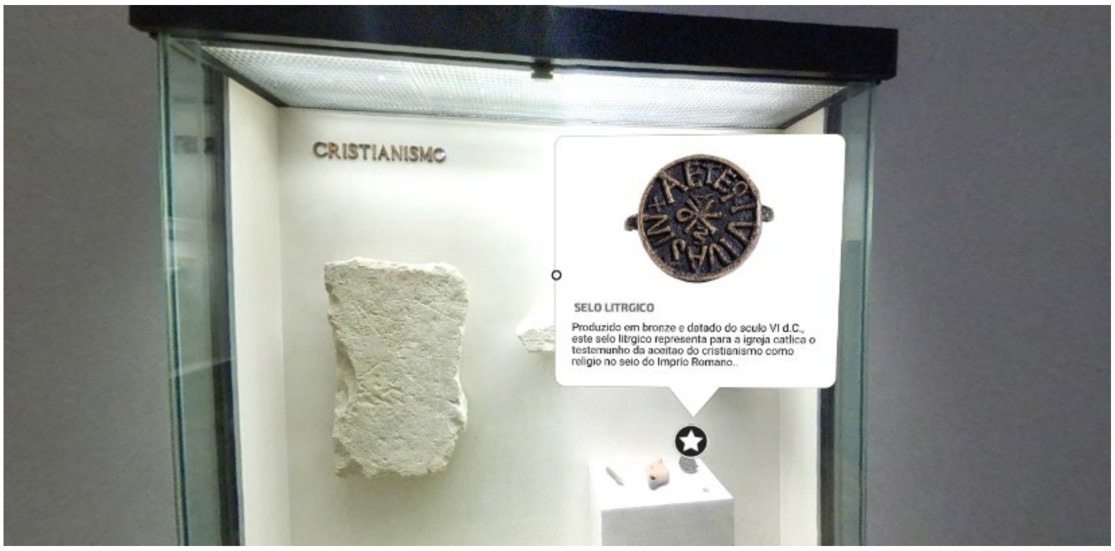
One of the goals of this project was to implement interactions with objects inside 360º virtual tours. In the context of the Conimbriga Museum, we opted to highlight the museum objects that are considered of national interest.

To make these objects salient, besides marking their locations with blue circles (as described before), we also highlighted the objects themselves. We experimented with two approaches for this: a white highlight around the object, and an identifying stamp (Figure 5).

The first approach was implemented by editing the panoramic 360º image with a photo editing software and drawing the white contours around the objects. The second approach was accomplished by inserting an <a-image> directly over the object in the 3D scene.

The next step was creating the information panels about each object to be seen when clicking on an object. Each information panel is composed by a background plane, an image, title and description (Figure 6).

For the panel to open when users click on an object, we positioned an invisible box in front of each object. This way, when users click on one of the national interest pieces, they are actually clicking on this <a-box>.



SELO LITRGICO

Produzido em bronze e datado do sculo VI d.C., este selo litrgico representa para a igreja catlica o testemunho da aceitao do cristianismo como religio no seio do Imprio Romano..

Figure 6: Information pop-up of an object.

To open the information panel, we created an A-Frame component that detects the click and performs a fade-in animation to make the panel visible:

```
image.setAttribute("animation__opacity", "property: opacity; dur: 800; to: 1;");
```

In this example, we change the opacity of the object to "1" in an animation that takes 800 ms.

For the animation to be repeated, we remove it before assigning it again:

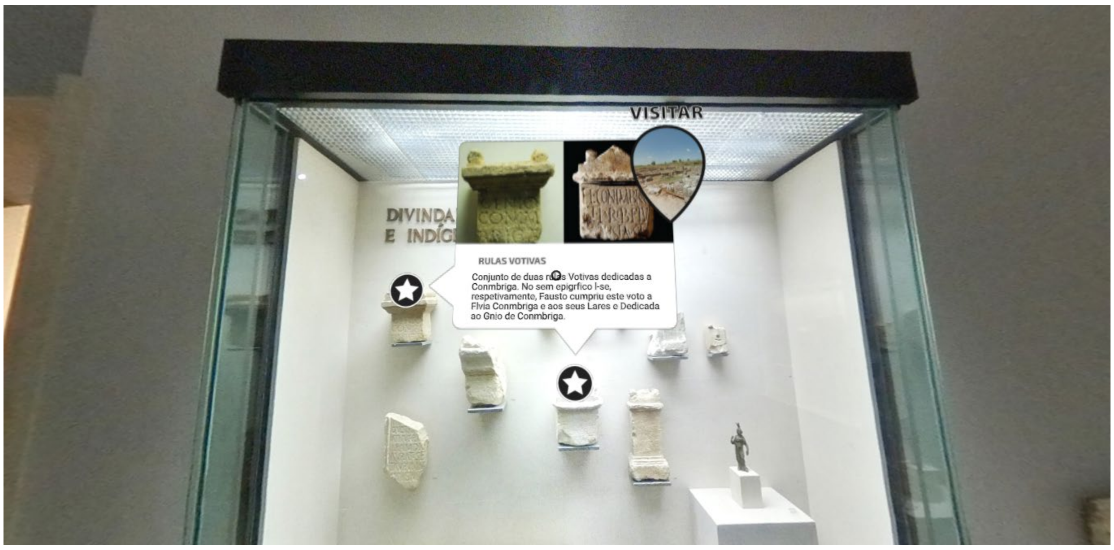
```
image.removeAttribute("animation__opacity");
```

After the information panel is opened, there must be a way to close it. Our solution is to automatically close the panel when the user looks away from it. To this end, the component we created also calculates the camera angle to the panel. If the angle is greater than a pre-defined value, we make the panel invisible. This way, users can read information about an object and just look away to dismiss the information.

## Portals

The last interactive elements of the virtual tour were portals to the outside ruins of the roman city, where the objects in exhibition have been found. Some of the national interest pieces have known precise origins in the ruins. For these, the virtual tour allows users to teleport to the outside location where the object was found.

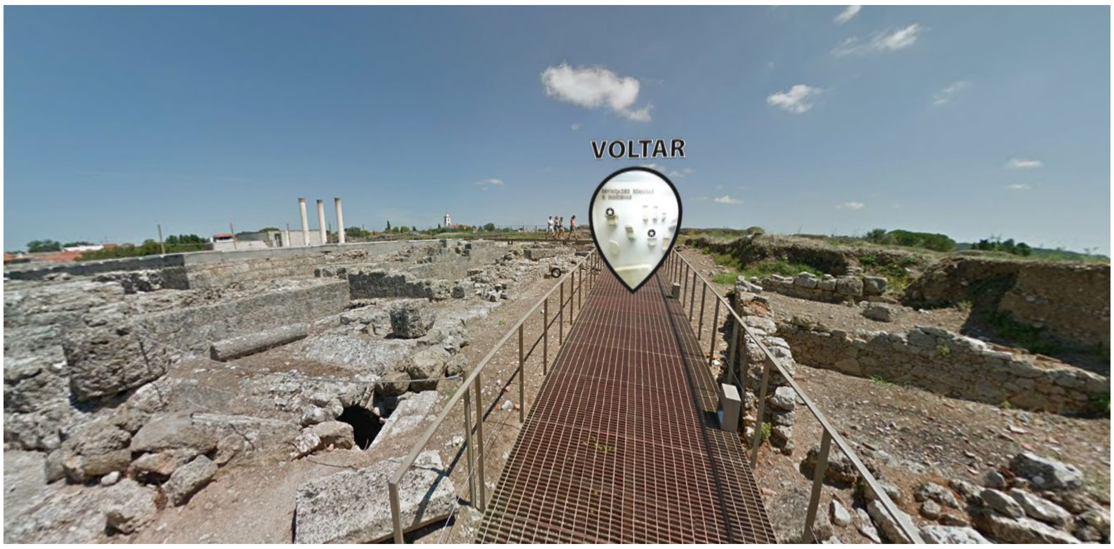
To this end, a portal was added to the information panel of these objects. The portal is positioned in the top right corner of the panel so as to be perceptible but not intrusive (Figure 7a). Implementing these portals was as simple as any other element of the information panel. A portal consists of an <a-link> with an image inside. In the outside scene there is also an information panel and a portal back to the museum. This allows users to read about an object in the museum and also to directly see where it came from, enriching the visit's experience.



RULAS VOTIVAS

Conjunto de duas rulas Votivas dedicadas a Conmbriga. No sem epigrfico l-se, respetivamente, Fausto cumpriu este voto a Flvia Conmbriga e aos seus Lares e Dedicada ao Gnio de Conmbriga.

(a) Portal within the pop-up of the object



(b) Portal on the site of the ruins

Figure 7: Portals inside (a) and outside (b) of the museum.

## MULTI-PLATFORM ADAPTATION

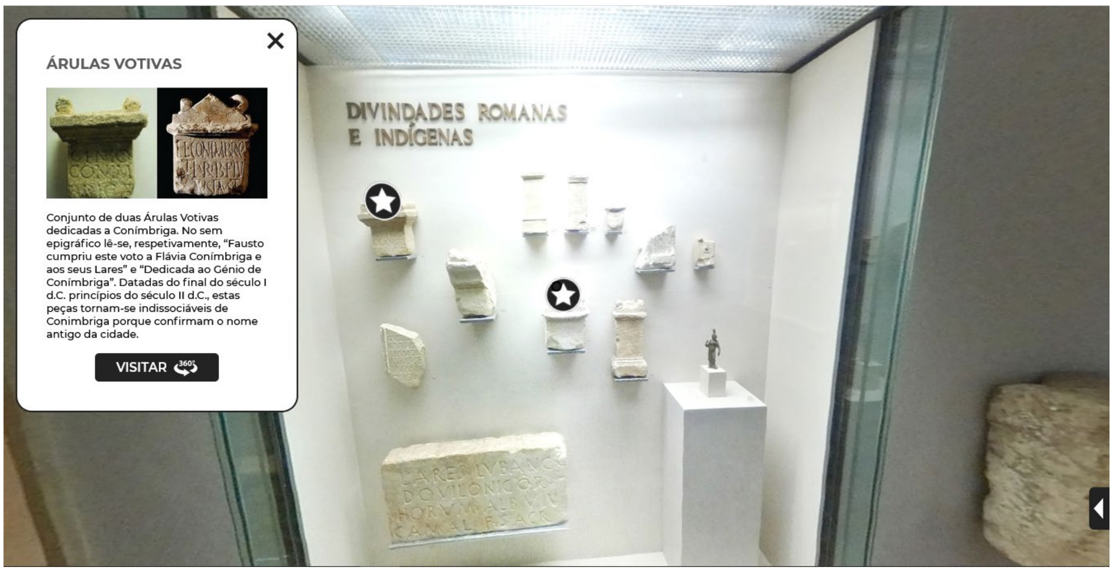
One of the aims of this project was to evaluate the effort of producing a multi-platform virtual tour, i.e., a virtual tour that could be experienced through desktop, mobile VR, and VR headset with controllers.

For the development, we took a mobile-first approach: all interactions were gaze-based and did not required any controller. Only a smartphone enclosure headset was required. We then adapted the implementation to desktop and then to VR headset with controllers.

### Desktop

The desktop adaptation has two main differences for the mobile version:

- Interaction is accomplished through the mouse

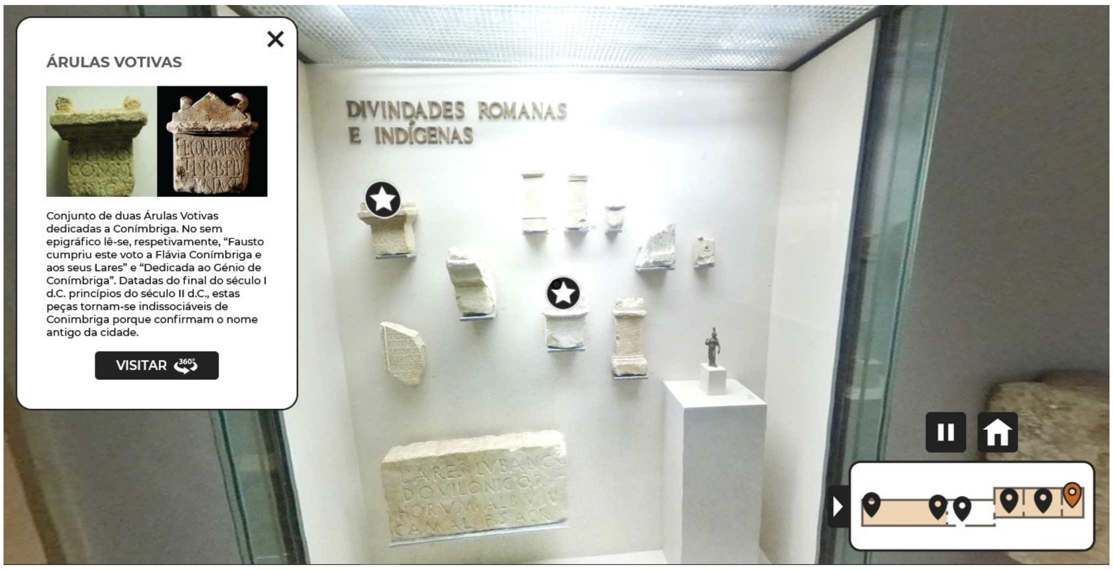


ÁRULAS VOTIVAS

Conjunto de duas Árulas Votivas dedicadas a Conímbriga. No sem epigráfico lê-se, respetivamente, “Fausto cumpriu este voto a Flávia Conímbriga e aos seus Lares” e “Dedicada ao Génio de Conímbriga”. Datadas do final do século I d.C. princípios do século II d.C., estas peças tornam-se indissociáveis de Conimbriga porque confirmam o nome antigo da cidade.

VISITAR

(a) Desktop interface with hidden menu



ÁRULAS VOTIVAS

Conjunto de duas Árulas Votivas dedicadas a Conímbriga. No sem epigráfico lê-se, respetivamente, “Fausto cumpriu este voto a Flávia Conímbriga e aos seus Lares” e “Dedicada ao Génio de Conímbriga”. Datadas do final do século I d.C. princípios do século II d.C., estas peças tornam-se indissociáveis de Conimbriga porque confirmam o nome antigo da cidade.

VISITAR

(b) Desktop interface with visible menu

Figure 8: Desktop pop-up and menu interface.

- Information panels and menus are presented in standard HTML instead of inside a 3D scene

Interaction through the computer mouse means that users can rotate the view by clicking and dragging and they can click on objects directly with the mouse cursor instead of having to use the center gaze cursor. This is mostly provided by A-Frame “for free”. Allowing users to click on objects using the mouse cursor instead of the gaze cursor requires us only to add the following attribute to the <a-scene> element:

```
cursor="rayOrigin: mouse"
```

Presenting information panels and menus in standard 2D HTML required rewriting the structure of the scene. These elements were removed from the <a-scene> and inserted as regular HTML (using <div>, <img>, <p>, <a>, and other elements) outside the <a-scene>. This is something were A-Frame does not provide any help. In addition, a CSS style sheet was added to style those HTML elements.

When an object inside the 3D scene is clicked, an information panel opens up in the top left corner of the window. These panels have a close button (a cross in the top right corner). If the panel contained a portal to the outside ruins, this portal is replaced with a button inside the panel, after the textual description of the object.

The navigation menu is now positioned in the lower right corner of the window and is hidden by default. To open/close the menu, users can click on the arrow at the left of the menu (Figure 8).

### HTC Vive with controllers

For the headset with controllers adaption we tested with an HTC Vive headset and associated controllers. The main differences to the mobile version are:

- Interaction with objects is made by pointing and clicking with the controller
- The floor menu is activate with a controller button, instead of by looking down

Interaction by point and click with a controller is again offered by A-Frame, requiring only the addition of an entity with the “laser-controls” component:

```
<a-entity button-listener laser-controls="hand: right" hand-controls="right"></a-entity>
```

To activate the floor menu with the controller buttons, we added a component to the controller entity that detected the button press. We called this component “button-listener”

## DIFICULTIES/PROBLEMS

The following issues were found during the development process. These are presented from the perspective of a Web developer, accustomed to developing websites with HTML and CSS.

### Links/Portals

Portals to other VEs are implemented with a regular HTML link element (<a>) which seems natural for a Web designer. Browsers treat portals just like they treat regular links in webpages: the current tab/window is unloaded and the new HTML document is fetched, parsed, and a new JavaScript execution context is created. This results in a clear break in the user’s presence: during a few seconds before the new VE is fully loaded and rendered the user sees a blank world. In some browsers/versions there is an additional issue: when traversing a portal, the browser will exit VR mode (full screen window with double perspective) and will not enter again automatically when the new VE is loaded, requiring users to explicitly press a button on the screen to enter VR mode again.

With traditional websites, developers can reduce the transition time between “pages” by creating so called “single-page applications” or websites that are fully loaded when accessed but display only a portion of the full content and disclose the rest of the content as the user interacts. A similar approach could be taken with A-Frame where the complete tour would be loaded so that traversing a portal would be a matter of instantaneously moving the user to the location of another 360° scene. This, however, might make loading times prohibitively large (unless a progressive loading mechanism is used), and would still not solve the issue for portals to VEs from third parties.

## Positioning Elements

Another issue is positioning 3D objects in a scene, for example positioning the arrows or circles in the “floor” to act as portals to other scenes. Currently, this is mostly a trial and error process. Although A-Frame provides a visual inspector with 3D capabilities for translating, rotating, and scaling objects in the scene, which greatly facilitates this task, its current version lacks power and flexibility. For example, most elements in a 360° scene must be positioned near the inside face of a sphere with a default radius of 5000 m. However, the 3D visual inspector makes it very hard to zoom in or out this far, making it cumbersome to adjust the 3D objects. In addition, it is necessary to go back to the HTML file to correct the values for the 3D position, rotation, and scaling, after adjusting them visually. The issue here is not just the 3D inspector tool – which seems mandatory for VR development – but the lack of declarative rules for positioning elements in a similar way to what Web developers are used to use in CSS, such as positioning elements relative to their containers. A-Frame takes the traditional 3D scene graph approach where each node creates its own coordinate space and where children are placed in absolute coordinates.

## Visually Complex Elements

Although A-Frame is based on HTML, it does not support any form of CSS. All styling must be accomplished by composing simpler elements, often requiring image editing for simple effects – an example is creating an information panel or floor plan with rounded corners (Figures 1b and 1c). This is very different from the typical styling process of an HTML document through CSS. The lack of declarative rules for setting dimensions, relative positioning, and basic styles such as borders and backgrounds makes it harder to test different visual solutions as one often does with traditional webpages.

## Text

In our virtual tour, text was mainly used in the information panels (Figure 1c). In many situations in traditional webpages, text is placed inside a non-dimensioned container so that the length and size of the text itself will dictate how large the container should be. In A-Frame, that is currently not possible: the text node must specify the size of the box that contains the text. This means that each information panel must be individually crafted (because the length of the information text varies). In one version of the tour, we actually ended up creating the information panel in an image editor and exporting the complete panel as a single image. This way of working is in stark contrast with typical web development, were HTML elements can grow or shrink according to their (current) contents. This is particularly relevant if text is loaded dynamically.

## Complex Interactions and Animation

Although there are various external components for animations and interactions, many situations still require creating custom components and, hence, JavaScript programming. An example of this is the floor plan that appears when users look down. The menu cannot be statically positioned because it must appear in the direction the user is facing, however, after it appears it should be static so that users can select a location (Figure 1b). This is an example of a very specific behaviour which required developing a custom A-Frame component. However, other behaviours/animations present in our virtual tour also required custom components: showing and dismissing the information panels, for example. Trying to implement these behaviours with the available animation components would result in unnecessarily complex set of declarations in the HTML (using the animation component, means that the animation rules must be written inline as the value of the attribute that represents the component, as one does with inline CSS, for example). The (now mostly) standard way to create animations in webpages is through the use of CSS transitions and/or CSS animations, where the animation rules are separate from the HTML they operate on. In addition, CSS provides pseudo-class selectors for simple interactions such as hovering or clicking on an element which makes it easier to declaratively set different visual styles. Something similar could be implemented for A-Frame, with perhaps specific pseudo-classes targeted at common VR interactions.

### Multi-platform support

One of our aims was to develop a web-based multi-platform virtual tour, i.e., a tour that could be experienced through a desktop computer, but also through mobile VR headsets, or fully-fledged VR Head-Mounted Display (HMD) with associated controllers (HTC Vive). Our approach was to start with a mobile version first and then make the necessary adjustments for desktop and HTC Vive. A-Frame makes it rather easy to do most of the adaptation because automatically supports mouse events directly on 3D objects (no need for a desktop user to place the centre screen cursor over the 3D object), and various types of controllers (a single declaration causes the VE to use "laser" controller that triggers events over the objects intersected by the laser). However, visual changes for different platforms are not so easily done. For example, on a desktop, it makes little sense to use a floor plan that is visible only when users look down. Instead it would make more sense to place the floor plan at the side of the screen always visible (or dismissable, like a side bar menu). To implement something as depicted in Figure 1d, we were forced to duplicate the existing floor plan code and create an HTML overlay over the 3D scene. The same for the information panel about an item: on a desktop, it makes sense to let users copy and paste the description text and associated images. In order to provide these interactions, we have to duplicate the contents and implement an HTML overlay. On a standard webpage, different layouts would be created by adding specific CSS rules together with media queries (sets of rules applicable only to specific classes of devices), but without duplicating the content. The issue here is greater than media queries to change the appearance of a given part of the interface. Different platforms have different capabilities and Web developers are used to thinking about and optimising websites for each platform (for example, by providing lower resolution images for mobile users). This is currently not easily done in A-Frame.

## CONCLUSION

Virtual Reality is not a novel concept or technology, but it has recently gained a new breath with the development of affordable consumer devices such as the Oculus Rift. It has also become possible to create Web-based VR experiences using simple declarative languages, but it is not yet clear how well adjusted to these languages are to the skills of Web developers.

In this work we aimed at analysing the development effort imposed by A-Frame – a Web-based VR framework. We proceeded by studying A-Frame and implemented a virtual tour based on 360º photographs of the Conimbriga Museum. The results show that, although A-Frame is certainly a low-entry barrier framework for VR development, there are still various issues that make VR development different from general Web development.

The ultimate questions this line of work aims to answer are whether a similar development strategy as the one used for Web development can be used for VR, and whether this is desirable. At first sight, it seems advantageous to be able to transfer one's knowledge from Web development into a different medium.

## ACKNOWLEDGEMENTS

This work was financed by national funds through the FCT - Foundation for Science and Technology, I.P., within the scope of the project CISUC - UID/CEC/00326/2019

## REFERENCES

A-Frame. (2019). A-Frame. Retrieved March 22, 2019, from https://aframe.io/

Behr, J., Eschler, P., Jung, Y., & Zöllner, M. (2009). X3DOM. In *Proceedings of the 14th International Conference on 3D Web Technology - Web3D '09* (p. 127). New York, New York, USA: ACM Press. https://doi.org/10.1145/1559764.1559784

Epic Games. (2019). Unreal Engine. Retrieved March 22, 2019, from https://www.unrealengine.com/en-US/what-is-unreal-engine-4

Facebook Inc. (2019). React 360. Retrieved March 22, 2019, from https://facebook.github.io/react-360/

Fraunhofer-Gesellschaft. (2019). X3DOM Documentation: Getting Started. Retrieved March 25, 2019, from https://doc.x3dom.org/gettingStarted/background/index.html

Unity Technologies. (2019). Unity 3D. Retrieved March 26, 2019, from https://unity3d.com/