Augmented Reality Proof Of Concept in Unity

An Android-based Decorator app. connected to the Cloud

Javier Rico(✉), Ricardo Luque, Anne-Claire Fouchier, Jingwen Yang
jvirico@gmail.com

I. Introduction

The wide spread of mobile augmented reality applications has shown the feasibility of real-time applications such as room decoration, by projecting 3D furniture models into a real scene.

The Decorator is an augmented reality app built on Android's ARCore in which one can virtually place multiple true-to-scale 3D object models in a real environment using a mobile phone. In this way, a user is able to, for example, furnish an empty room easily by choosing different objects from a cloud-based catalog, and instantly have a direct sense of the overall scene. We use several pieces of furniture in this demo project.

The present use case allows to furnish a whole room in just a few taps. The decorator is an exciting way to enjoy designing, an easy way to create ideas, and it is easily extensible to other use cases for fast creation of virtual prototypes on top of existing real spaces, by simply uploading new objects to the cloud-based catalog of 3D objects.

II. Project Plan

This project has been developed under a SCRUM methodology, several iterations in the form of Sprints produced the shown tasks distributed along 4 months as in the Gantt diagram of Fig. 1. We list below a description of the tasks that comprise the plan followed to develop the final application.

1) **Training**: Self-training with study materials available on Coursera [4] and YouTube [1].
2) **Scope**: Narrow down the scope of our project. Draft the general schedule to tackle the challenges presented.
3) **Set up**: Set up the development environment with tools such as Unity Hub and Github repository.
4) **Project Baseline**: release of version 1.0 of the application, which included a basic User Interface (UI) composed of a few static buttons stored in the application.
5) **Object Library**: release of version 1.0 of a local library containing a base of three 3D furniture models.
6) **User Interface**: release of version 2.0 of the application. User Interface is improved by replacing static buttons with a sliding bar, see Fig. 4.
7) **Library expansion**: update of the local library to version 2.0 with six 3D furniture models.
8) **Usability**: release of version 3.0 of the application. Crosshair-based interaction using a marker is implemented to improve usability on phone and tablet devices.
9) **Documentation**: Report drafting process was launched.
10) **Library update**: library updated to version 3.0. The local database is replaced by a cloud database.
11) **Architectural improvement**: Release of version 4.0 of the application. Objects from Cloud based library are automatically imported and displayed in the sliding bar, Fig. 4.
12) **UI improvement and final release**: release of version 5.0 of the application. Users are able to change the scalability and rotation of loaded objects with a new control menu.

III. Library

The 3D furniture models used in the **Decorator** app are obtained from Sketchfab [3]. Sketchfab is a platform to discover, share, publish, sell and buy 3D, VR and AR content. It provides a viewer based on the WebGL and WebXR technologies that allows users to display 3D models on the web, to be viewed on any mobile browser, desktop browser or Virtual Reality headset. As shown in Fig. 2, our 3D furniture models are common appliances that are used in a bedroom, such as chairs, beds, tables, and sofas. This is because the aim of the project is to furnish, refurnish or rearrange a room in an augmented reality environment using a mobile device.

IV. Implementation

This project is mainly implemented in C# language. Following the design patterns and methods employed in its implementation are listed:

- **Singleton:** Singleton is a design pattern used in object-oriented programming to permit no more than one instance of a class.
  In this project, the Singleton design pattern is used for DataHandler and UIManager classes. In the DataHandler class it is used to ensure that each object in the augmented reality scene is assigned to one instance only, to keep the instances exclusive from one object to another and not run into any instancing problem. DataHandler is used to specify the behavior of the buttons corresponding to the object **interaction panel**

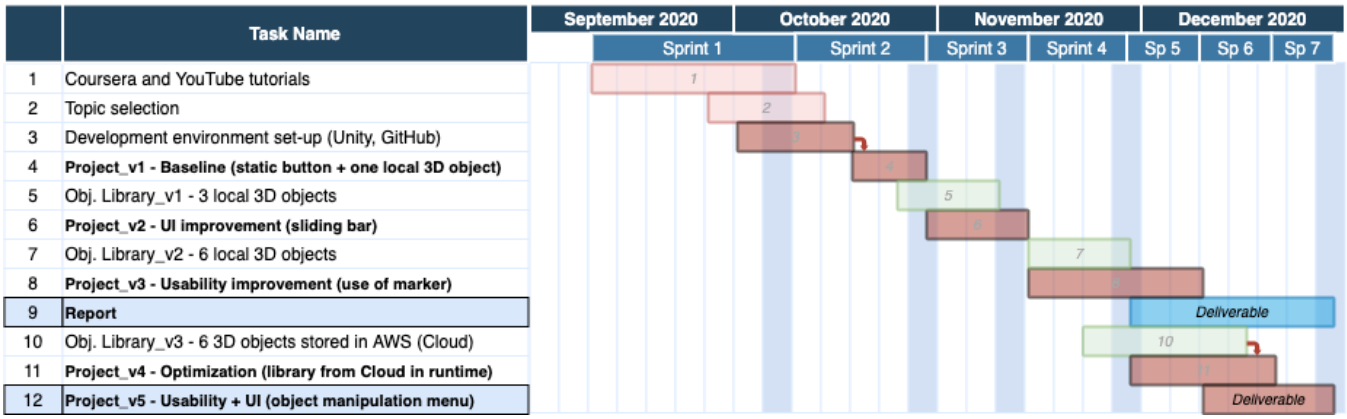

Fig. 1: Gantt Diagram

Fig. 2: Furniture examples from Sketchfab

(seen in Fig. 3) during runtime. It also creates and manages an instance for every furniture piece in the scene. In short, it is in charge of *initializing* the behavior of every augmented object, and every other button in the interaction panel.
Similarly, in the UIManager script, the **Singleton** pattern is used to ensure every element, or button, within the slide bar has its own instance. It is used to manage the scrolling bar which displays the available furniture stored in the cloud database. The scrolling or sliding bar can be seen next in Fig. 4

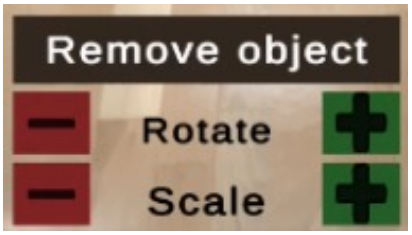

Fig. 3: Interaction panel

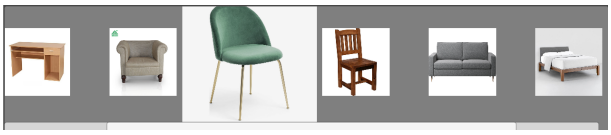

Fig. 4: Scrolling/Sliding bar

- **ScriptableObjects:** ScriptableObject is a class that can be derived from to create objects that do not need to be attached to game objects. It is mostly used as a data container to save large amount of data, independent of class instances [5]. It is usually used in order to decrease memory usage so that copies of values or objects are not created in memory.
  In this project, it is used for creating multiple pieces of the same object on the scene, so that every time a Prefab is intantiated it gets its own copy of data and access to it by referencing the Prefab. In other words, there is just one copy of the data in memory, instead of having a copy per each furniture piece.

- **Addressable Asset System:** The Addressable Asset System makes it possible for the developer to ask for an asset via its address. It generates an address which can be called from anywhere once an asset (e.g. a prefab) is marked ”addressable”. No matter where the asset resides (local or remote), the system will locate it and its dependencies, then return it [5].
  As a good architectural design practice, in this project we make use of *addressables* by storing resources

outside the application, both the 3D object models and their corresponding images displayed in the User Interface, are stored in an Amazon Web Service Bucket, see Fig. 8.
Storing information in the cloud avoids overhead in the application as well as its increment in size. No only so, but it allows updates or modifications in the object library without the need of application updates or builds. When developing a mobile app, its size is a crucial factor, since mobiles usually run into storage limitations. Online storage basically reduces the burden on the phones' hard disk and improving each object accessibility.

- **Cross Hair based interaction (marker):** Cross Hair interaction defines a way in which the user can interact with a game or application by defining and displaying the position of the cursor. The cursor makes it easier for a user to know where an object will be placed.
In this project, we place the cross-hair cursor in the middle of the screen of our application, on the precise location where we want to place an object. By doing that, we avoid using finger to touch the screen every time and hence improving the visual experience. It is an alternative of finger interaction.

V. User Interface

The User Interface has been changed multiple times with incremental improvements. Following, the 5 versions of the user interface are described.

1) **Release 1.0:** Presented only static buttons to choose the furnitures. This presented several problems since objects could only be added to the scene, yet not removed. The objects were not rotated nor scaled, a poor plane detection was performed and the planes were visually perturbing since they were drawn on the real world scenario. A snip of its behavior can be seen next in Fig. 5.

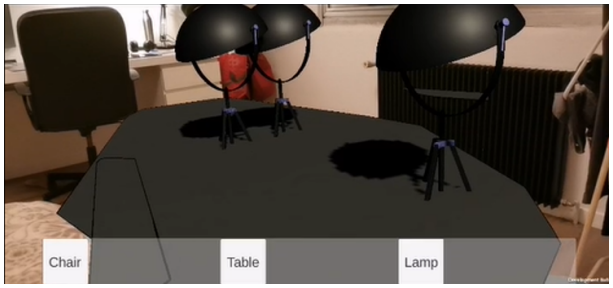

Fig. 5: Version 1

2) **Release 2.0:** Replaced static buttons with a scrolling bar with multiple sliding buttons (objects stored inside locally).

3) **Release 3.0:** Improved the scrolling bar with an animation effect [2]. We added zoomimg in and zoomimg out visual effect for the button in the center to make it clear which piece of object is now being chosen. Its behavior can be seen next in Fig. 6. The detected plane is made invisible to create a more immersive sensation. Moreover, a Cross hair based marker is introduced to replace the detected plane. In here, the marker is directly on the detected plane as seen in Fig. 7. The behavior can be seen in the video explaining the app functionalities.

Fig. 6: Version 3

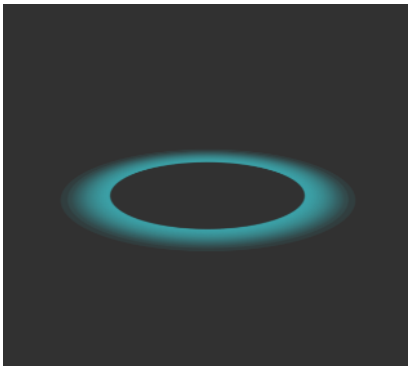

Fig. 7: Version 3: Marker

4) **Release 4.0:** An automatic synchronization of buttons in scrolling bar in User Interface from object models that are stored in a Bucket in Amazon Web Services as displayed in Fig. 8. Each object is compiled in one .json file and one .hash file. A reference of how some of the objects are loaded including a corresponding image to identify each button can be seen in Fig. 4.

5) **Release 5.0:** An additional button panel (Fig. 3) was added in the user interface to manipulate the scaling, rotation and deletion of objects.

The finalized user interface is displayed in Fig. 9. The blue circle in the center results from the Crosshair based interaction. The Sofa is the object displayed by choosing

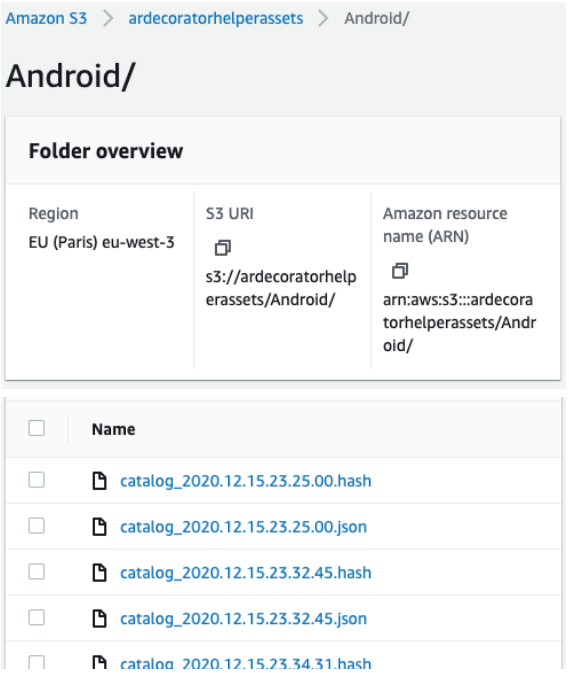

Fig. 8: AWS Cloud Database

the central object in the scrolling bar at the bottom of the image. We can see there are other objects not chosen by the user display on both sides of the scrolling bar. Apart from that, the button panel has three functionalities. The top one serves for the removal of the object. The other two respectively works for rotating and scaling the object.

VI. Functionalities

As we have seen in the UI case, new functionalities have been added withing each iteration. Following, some specifics of the functionalities implemented in intermediate stages of the project. The final deliverable can be reached in the following repository[1], whereas an example of its behavior can be found here[2].

1) **Release 1.0:** The objects are placed in the scene by touching the position on the screen where you want to place it. The localization is not very precise.

2) **Release 2.0:** Replaced static buttons with a scrolling bar with multiple sliding buttons (displaying objects inside the database).

3) **Release 3.0:** Crosshair based interaction is utilized to improve positioning accuracy. A circle marker is used to place the object, in order to facilitate precision. This technique is very useful when using the app in tablets for example, where holding the device with one hand is uncomfortable.

4) **Release 4.0:** The selected object is highlighted. We can manually adjust their situation in the scene. One tap outside the object unselects it.

5) **Release 5.0:** Users are able to scale up and down, rotate, and delete any of the objects. This functionality substantially improves user experience.

Fig. 9: User Interface

VII. Future Work and Conclusion

On one hand, the implementation allows a rapid behavior of the app. Furthermore, addressables make it easy to reduce the hard disk phone storage, making the app suitable for database update without having to modify the app. Cross-hair cursor facilitates the interaction between the real world and the augmented reality objects.

[1]Code available at https://github.com/jvirico/augmented-reality-poc-unity
[2]Example videos https://github.com/jvirico/augmented-reality-poc-unity/tree/master/vid

On the other hand, handling occlusions in the horizontal cross-section direction was not implemented due to the need of using vertical planes and detecting which vertical plane the introduced object is colliding with. A simpler approach to handling vertical cross-section direction is implemented by grounding the element to a plane, i.e. making sure that the vertical difference between the object's base and the plane is zero. Also, although full functionality of the mentioned user interface is implemented, the scope of this prototype leaves for future efforts a more professional-looking user interface, since the current state has an 'under construction' visual impression.

Regarding the cloud-based library of objects, richer metadata associated to 3D objects, such as information about color, brand, price, could be incorporated to extend functionality. Improvements in relation to informing the object sizes in real scale may be also a desirable feature, specially after the scaling up and down functionality.
Finally, functionalities to save the favorite products per user could be developed, and interfacing with social networks by for example, sharing the selections of items on social platforms, or facilitate purchases directly through the app could extend the project further.

REFERENCES

[1] J. Lab, *Unity ar foundation tutorial*, [Online; accessed 10-November-2020], 2008. [Online]. Available: https://www.youtube.com/c/JoystickLab.

[2] DOTween, *External plugin DOTween documentation.* http://dotween.demigiant.com/documentation.php, [Online; accessed 19-November-2020], 2020.

[3] Sketchfab, *Sketchfab VR and AR models discovery platform.* https://sketchfab.com/, [Online; accessed 1-October-2020], 2020.

[4] C. Training, *Handheld AR App Development with Unity.* https://www.coursera.org/learn/handheld-ar/home/welcome, [Online; accessed 21-September-2020].

[5] Unity, *Unity official documentation (script reference).* https://docs.unity3d.com/ScriptReference/, [Online; accessed 1-October-2020].