Persistent Placement Platform for Interactive Objects in Augmented Reality

Shaon Rahman*,
University of Stavanger, Norway
Email: *s.rahman@stud.uis.no

***Abstract*—Augmented Reality is an interactive experience of the real world, enhanced by placing computer generated objects inside it. In this project, we developed a cross-platform Augmented Reality application that can place and manipulate objects inside the Augmented Reality environment. The coding for the application was done with modularity, extensibility, and reusability in mind. The same codebase can be forked to develop more customized and specialized Augmented Reality applications.**

***Index Terms*—Augmented Reality, AR Foundation, Unity**

I. Introduction

An Augmented Reality (AR) environment places virtual objects into the real world. AR exist on a spectrum called "Virtuality Continuum" which gives different names to the level of the melding of virtual and real worlds.1. The left side of this spectrum points to the real world without any enhancements where the right shows the opposite extreme in which the full environment is made of virtual objects.[1].

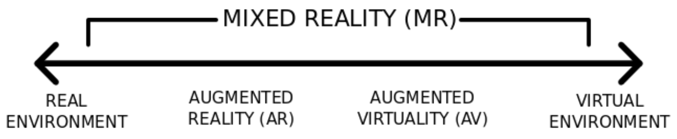

Figure 1. Virtuality Continuum according to [1]

The AR environment, helps the user to see the physical world, with simulated objects superimposed on or composited with the real world. Therefore, AR complements truth rather than replace it entirely as opposed to virtual reality. Ideally, it would seem to the user that virtual and physical objects coexist in the same plane of existence.[2].

The AR is useful for visualization of complex 3D systems such as mechanical devices[3], [4], [5], human organs [6], [7], [8] and architecture or buildings[9], [10]. It is more intuitive than a screen because a screen displays 3D information on a 2D surface whereas AR places objects in the 3D environment in the context of the real-world objects around us. It can be extremely helpful in education[11]. For example, seeing an animal in front of us gives a better perspective on the actual size and shape of the animal and the user can watch it from any angle which can be a great learning experience. The same method can be used to train medical professionals to identify and better understand the inner workings of humans [6], [7], [8]. AR is an interactable environment, so we can also modify and manipulate the objects we see inside it. This allows the users to not only understand how complex systems in our real-world work but also design and develop new systems. AR can be used in the automotive industry to design new engines and components [12], [13], in architecture to help design bridges and new buildings [9], [10] and simulate how the newly designed models work and better understand how and where they fail and reiterate without leaving the AR environment.

Some of the currently available AR technologies are Microsoft Hololens and Magic Leap 1. These are the leading head-mounted AR displays at present. Most modern-day smartphones are also capable of projecting AR content although their competence varies depending on the hardware available. Smartphones with depth camera or for the best AR capability, LIDAR sensors work better than the general smartphones without these hardware[14]. More handheld devices are slowly adopting new AR capable hardware [15].

AR is currently limited by the competence of the device's tracking and gesture recognition system. Head-mounted AR displays rely on finger tracking and hand gesture recognition to take inputs from the user. Although these systems work very well they are susceptible to glitches which can make the workflow a bit annoying [16]. Handheld devices such as smartphones are not particularly made for AR use-cases which leads to working in AR with handheld devices frustrating. The smartphone is needed to be held in one hand which greatly limits the usability of the AR environment. Smartphones also have small screens which result in the user always looking through a small window to see the AR environment which is in stark contrast with head-mounted devices. Head-mounted devices can deliver an immersive experience as it can completely surround our vision, and it can also track our gaze to interact with the AR environment. Smartphones have very low computational power compared to head-mounted devices (powered by a dedicated computer) which limits their capability of realistically rendering and simulating complex 3D objects. The vast differences between these two fundamentally different platforms can also make developing applications for them difficult since their inputs and capabilities are very different.

Our task in this project was to create a modular and easily extendable application with a codebase that is highly reusable. The application covers the most common Human AR interactions such as placing, moving, and manipulating objects in AR in a way that is platform and AR tracking system agnostic. We break down the codebase into multiple small pieces of code with single concerns.

We used Unity's real-time 3D engine and its AR foundation framework, so we can target head-mounted devices and smartphones alike. Unity is a very high-performance 3D engine with cross-platform support. Unity is used for game development[17], animation and storyboarding[18], simulation[19], architectural visualization[20], etc. It is in rapid development with a highly active community. It ships with full .Net framework and C# which together create a very mature platform with stable libraries and packages. AR Foundation framework is a Unity package that can be used to detect objects in the real world. It can detect horizontal and vertical planes, human poses, faces, point clouds, images, 3D objects among other things. These are called 'trackables'. AR Foundation is also cross-platform that can target Androids, iPhones, and Hololens as build targets. Its competitors ARkit, ARCore, Hololens SDK on the other hand only support iPhone, Android, and Hololens respectively.

In the current implementation, we have only used horizontal plane detection (also known as ground plane detection) for placement of virtual objects. However, with minor modifications to the codebase, any object that the AR Foundation framework can detect could be used to place AR Objects. The application consists of the following parts.

- User Interface (UI)
- Asset Inventory
- Object Manipulation System
- Scene saving and loading System

The rest of this report is organized as follows. UI design describes how the user interface of the application is designed, how users can interact with the application. Implementation describes the inner working of the application. We talk about how user interactions are handled in the code and how the application reacts to user actions. We describe each component of the application and detail what each component does and how the components interact with other components. In the Results and Discussion section, we discuss the outcome of the project and highlight the current limitations of the AR application development process. In the Conclusions and Future Work section, we talk about how the application can be extended and what can be done in the future with new AR technologies that are emerging.

II. UI DESIGN

The code for the application was written to be easily extendable and modular in nature. The developed application is cross-platform and is built on top of the AR foundation framework. For this reason, we decided to limit the interactions with the application to be mostly based on UI elements (i.e. buttons, sliders). Apart from selecting/deselecting an object in an AR environment all the other operations such as scaling, repositioning, etc. are done by clicking buttons or using UI sliders. The selection and deselection are done by pointer click action on a desktop and touch screen on a mobile phone. We kept the functionalities UI based, instead of gesture-based so that the application can be built for head-mounted AR devices with minimal effort. The core design of UI depends on the following four states.

- Initial state
- Placement state
- Selection state
- Edit state

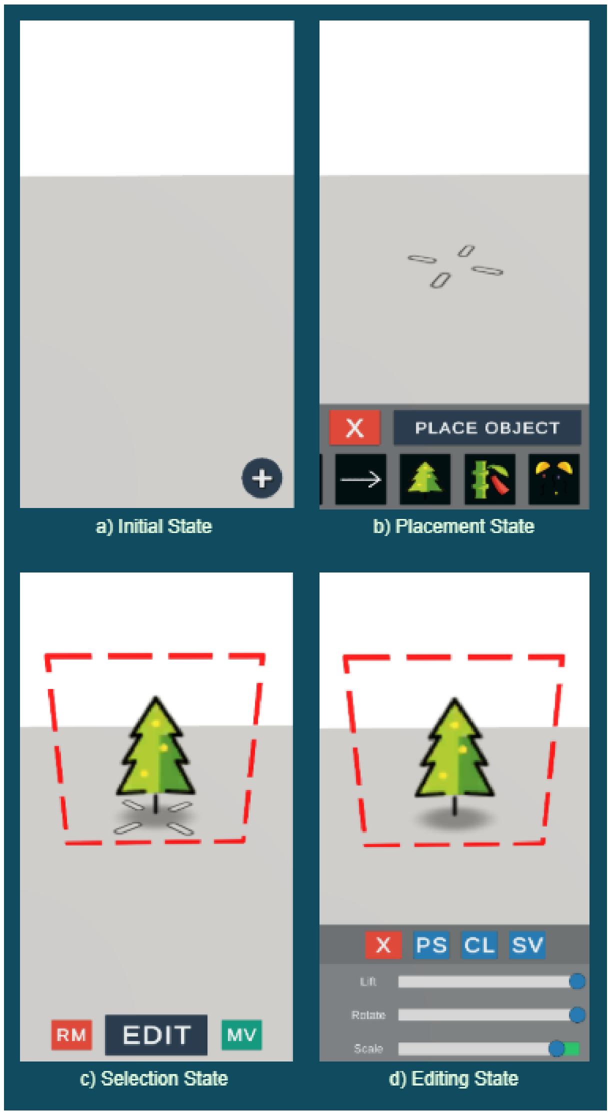

Figure 2. Interfaces on each UI state.

At any given time the user can be in one of the four states. The transition between the UI states can be triggered by a Button click, selection of an object, or deselection of an object. The buttons that contribute to UI state transitions are shown in Figure 3.

The program starts at the initial state. In this state, the user can have a clear view of the AR environment with minimal obstruction. The only UI element present at this view is the create button. Clicking this button transitions the UI from the Initial state to the Placement State. The Placement state is

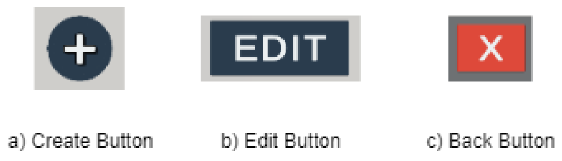

Figure 3. Buttons that triggers UI state change

used to place objects from UI to the AR environment. From any state, if an object is selected, the UI transitions to the Selection state. In this state, the object can be repositioned, deleted, or transitioned to the Edit state by clicking the Edit button. Clicking the Back button from any state will transition the UI to the Initial state. The flow of the UI is shown below.

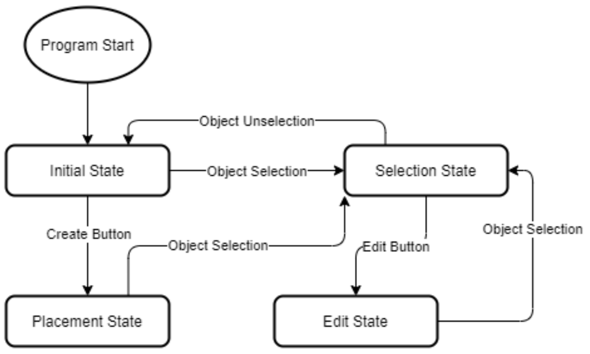

Figure 4. State Transition diagram for the UI.

In the placement state, a cursor is projected on the detected ground plane to mark where the new object will be instantiated. A library of objects is also shown in this state. The library is presented as a horizontally scrollable list of objects. The user can select an object from the list and click the Place Button to instantiate the object in place of the marker.

The state is accessed by selecting an object in the AR environment. Upon selection, the UI state is changed to Selection state and three buttons are shown. The selected object is highlighted with a red dashed line around the object. The left-most button deletes the selected object and the right-most button is used to move the object to the desired location marked by the cursor on the ground. The middle button changes the UI state to the Edit State.

In the Edit state, the cursor on the ground is hidden and a panel with three tabs are shown. In the first tab the user can use the three slides to lift the object off of the ground, rotate the object or scale the object to make it larger or smaller. The second panel can be used to change the color of the object using a color wheel. The third tab can be used to save the configuration of the objects in the AR environment into the disk or load a saved configuration.

III. Implementation

Unity loosely follows Entity Component System (ECS) design pattern. ECS is an architectural pattern that is mostly used in game development. ECS follows the composition over inheritance principle that allows greater flexibility in defining entities where every object in a game's scene is an entity (e.g. enemies, bullets, vehicles, buttons, healthbar, etc.). Every entity consists of one or more components that contain data or state. Therefore, the behavior of an entity can be changed at runtime by systems that add, remove, or mutate components. This eliminates the ambiguity problems of deep and wide inheritance hierarchies that are difficult to understand, maintain, and extend[21].

Unity does not follow the pure implementation of ECS where Entity, Component, and Systems are separate. Unity implements entity and, bundles components and systems together so every component system can be either a class with data and code together or just a data class that only holds the data and does not have any code. But, any class that can be assigned to an entity in Unity needs to inherit from the Monobehaviour base class which comes with the Unity package and is a closed source implementation. So a Component in Unity is a general C# class that inherits Monobehaviour base class. Monobehavour exposes many functions that are called in different scenarios. So even if a data class is created just to hold data it still receives function calls defined in Monobehaviour class because it is a subclass of Monobehaviour, which makes data classes in Unity very inefficient. Therefore, we try not to use data classes in Unity and stick with general classes that hold data and code. The code is written to be modular in nature and extendable where possible, so new functionalities and features can be implemented easily in the future. The components written for our application are described below.

The UI state and transitioning is implemented based on an event system. The UI elements in each state are grouped together and the whole group is controlled by the UIDisplayer component. This component is responsible for showing or hiding the group of UI it is assigned to. It stores a coordinate for showing position and another coordinate for hiding position. It has two public methods 'Show' and 'Hide'. When Show is called the UIDisplayer moves the group of UI to the showing position and calling the Hide takes it to off the screen where the user can not see the group of UI. Certain user actions such as button click or selection of an object trigger events and the Show and Hide function of particular UIDisplayer are called to transition from one UI state to another.

UIManager component holds the references of all the UIDisplayers present at the scene. It is responsible for calling the Show and Hide functions on the UIDisplayer component. It also triggers an event when the state of the UI is changed. It passes the name of the new state as an Enum named UIMode. UIManager exposes public methods for each of the UI states. And it directly listens to the button click events of the buttons or any other user actions that change the UI state. It manages all the state transitions.

ControlPanelManager component is written to manage tabbed layouts. This component is used to tab through the UI elements in the Edit state. The tabbed panel holds all the tabs one after another horizontally. When the user triggers a tab change event, the whole tab group is shifted by the width of the screen so the next set of UI elements can be shown. In our case, the button clicks on the Edit panel trigger the tab shift. It is written this way to facilitate new tabs in the future to extend the functionality of the application.

The AR objects which can be placed in the AR environment have a C# interface called IObjectControllable. It exposes the functionalities required by each of the objects such as select, deselect, lift, rotate, scale, move, delete. So all the AR objects can be referenced using this interface because of polymorphism. And it also forces implementing the same features in all object classes because a 2d image, a text, and a 3D object do not have the same implementation of the select method all of these classes implement their own method for selection. It also allows extendibility because we can add new type of objects in the future by inheriting this interface and implementing the functions defined in the interface and placement, selection logic will still work as they reference the IObjectControllable and not any specific class.

ARshapeObject and ARtextObject are two components that inherit IObjectControllable interface. Shape object is a component for any 2d image AR object, Text object is a component for AR text object. These two components work very similarly and only the implementations of each of the member functions are different. The Shape object has a red dashed line as a selection marker but the text object implements a red outline for indicating the selection. Both of the objects of the components are instantiated using the ShapeFactory component.

ShapeFactory component is a factory design pattern that creates 2D AR objects before placing them in the scene. It holds a reference of a list of all the images that can be initialized as a 2D AR object. The list of images is used to dynamically generate the buttons in the UI of the asset library. That same image is used to create a new 2D AR object when the button with the image is clicked in the asset library. Upon the button click the AR object is created with appropriate components and a reference is sent to the PlacementManager component. If the user clicks the text button instead, the ShapeFactory initializes a text object and displays a text input field using the UIDisplayer component. After the user finishes writing the text, the Shapefactory creates a text AR object and sends a reference to the PlacementManager as the object to place. The placement is not done until the place button is clicked. New images can be added to ShapeFactory just by adding a new image to the list of images. ShapeFactory automatically creates the button for the image, adds listener to the button, and creates the AR object and all required components when the user clicks the button.

PlacementManager component handles placing an object to the scene. When the public method for placing is called in the the component, a ray is cast from the middle of the screen in the direction of the screen normal. if the ray hits a ground detected by AR Foundation it gets the position of the ground where the ray intersects the ground and places the object if there is an object reference. The place method is called when the 'Place object' button is clicked. Any component can add an object to be placed in the scene using this component. Primarily, in our use case, only ShapeFactory class assigns objects to PlacementManager but It can be easily implemented so other classes or even objects in the scene also assigns objects to PlacementManager to be instantiated. So more dynamic objects can be easily made that itself creates new objects in the scene. Placement Manager also sets the currently placed object as the children of ARObjectRoot. ARObjectRoot is an invisible object which is the parent of all the objects initialized in the AR Environment. When the first object is placed in the scene, the ARObjectRoot's position is changed to the first object and then the first object and any object after that is set to be the children of the ARObjectRoot object. The parent object plays a crucial part in saving the configuration of objects.

SelectionManager Component holds the reference of currently selected object. The selection/deselction logic is also implemented in this component. An object is selected by tapping on the screen. When the user taps on the screen, a Ray is cast from the tap location towards the normal of the camera. If the Ray hits a IObjectControllable object we set it as the currently selected object and trigger the Selection event and pass the selected object as a parameter to the event. if the ray fails to hit any object with IObjectControllable interface we set the currently selected object to Null and trigger the deselection event. Any component can listen to this events to trigger some actions. For example, when an object is selected, the selection marker of that object is shown and hidden when it is deselected. The UI panels also synchronize with currently selected object so the object can be manipulated through the UI. The UI state is also changed when a selection event is triggered. We can add more actions just by listening to this events such as an animation can be played on the object when it is selected or a set of buttons or information can be shown about an object through a pop-up when it is selected.

SaveLoadManager saves all objects in the scene relative to the position of the ARObjectRoot object. So if the configuration is placed later in a different environment in a different ground location the relative positions of the configuration being loaded are kept consistent.

There are also some small scripts such as SelectionMarkerController which shows/hides the red selection marker around an object when selected/deselected. CrossHairControl which displays and moves the cursor on the ground when the UI state is in Selection or Placement state. SimpleSelfRotation script rotates an object on its own axis. It is used for continuously rotating the cursor on the ground. MoveRemoveSelectedObject is a simple script that handles the repositioning and removing of an object from the scene.

For the visual appeal, some animations were added to the object placement, removal, and UI state transitions. All these animations are interpolation-based so instead of objects and UI elements jumping from one place to another they cover the move to the destination slowly over a short duration.

IV. Results and Discussion

The results were as good as the device's capability of tracking the real world. When deploying the application on a handheld device without any depth or LIDAR sensor there were some tracking issues that were beyond the scope of the project as we were relying on Unity's AR Foundation framework and hardware for tracking. The saving feature did not work on the test device which was an Android smartphone but the saving system is tested to work on Desktop. The asset inventory and UI based object manipulation system work as intended. There are some inconsistencies when selecting an object projected on the AR scene when multiple objects are placed on top of each other or when objects are placed in a distance. That is because the rays that are cast from the phone screen to the AR scene would miss the small object boundaries or only select the first object when a lot of other objects are in front.

The biggest hindrance we faced while working with Unity and AR Foundation was that there was no testing framework for AR in the engine. The only way to test the code is to compile and deploy it in the test hardware and test from within the testing hardware unlike general smartphone or computer applications which can be tested right from the development machine. This process takes a lot of time to reiterate over minor adjustments or to fix bugs. Until an emulator is developed for testing AR applications in Unity this process will be very hard to work on. The problem is even more so for applications that depend on GPS locations along with real-world tracking because those applications need to be tested from different GPS locations. The same problem arises if the application is collaborative where multiple users can work in the same AR environment. It would require multiple persons to test the applications through the development process in current circumstances.

We originally wanted to target the application to track horizontal planes as well to make a board on a wall where images, texts, stickers, or sticky notes can be placed. In practice, it turned out the test hardware was having major issues detecting horizontal walls, and testing the application was getting very difficult. But we found that our hardware can detect a horizontal plane much better than a vertical plane. So we developed the application for horizontal planes and added lifting of objects above ground that way it still serves its original purpose.

The full source code can be found on Github (https://github.com/Wizdore/AR_Project). A pre-built android executable can also be found in the link. A demonstration video of the application (https://youtu.be/JjdaOWXR9-M). A sample photo of Text and 2D stickers is in Figure 5.

V. Conclusions and Future Work

In this project, we made an application that can place objects in the AR environment and modify various properties of the objects. The code is written to be reusable and modular by breaking the code down into individual functions.

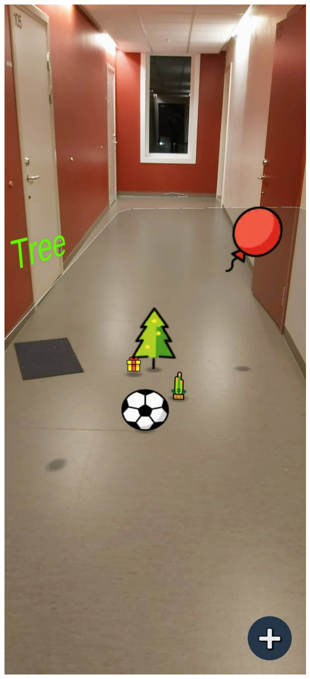

Figure 5. Screenshot of the application

The application can be further extended by adding location support with Google's cloud anchor feature which Google announced on 6th October 2020 when working on this project. The cloud anchor allows one person to place an object in a real-world location in the world and another person can at anytime see what the other person placed in the same location through the AR application. It is a cloud-based persistent spatial AR service. It can open up a lot of opportunities for AR, for example, AR-based multiplayer games, teaching system, or annotation system to display information about historical places or in advertisement and information system for shops and businesses.

Unity and AR Foundation framework are developed rapidly. Unity gets two major version releases every year, although the AR Foundation framework does not follow upgrade schedules, it also gets major releases every now and then. We started the project with the AR Foundation version 3 and while working on the project, version 4 of AR Foundation was released. New version releases include major new features such as AR Foundation 4 now has meshing support which generates 3D meshes from vertical, horizontal planes, and point clouds. It greatly improves the quality of tracking objects in AR environments and supports object occlusion which makes the AR environment more legible to the end-users. The rapid development also comes with a drawback. Major version releases might break compatibility of the code written for older versions therefore maintaining the codebase for AR applications can be very difficult. With new and better technology on the edge, the codebase needs to be up to date to provide the latest and the greatest features.

References

[1] P. Milgram and F. Kishino, "A taxonomy of mixed reality visual displays," IEICE Trans. Information Systems, vol. vol. E77-D, no. 12, pp. 1321–1329, 12 1994.

[2] R. T. Azuma, "A survey of augmented reality," Presence: Teleoperators & Virtual Environments, vol. 6, no. 4, pp. 355–385, 1997.

[3] D. Scaravetti and D. Doroszewski, "Augmented reality experiment in higher education, for complex system appropriation in mechanical design," Procedia CIRP, vol. 84, pp. 197–202, 2019.

[4] Y. Wang, S. Zhang, S. Yang, W. He, and X. Bai, "Mechanical assembly assistance using marker-less augmented reality system," Assembly Automation, 2018.

[5] M. Fiorentino, A. E. Uva, M. Gattullo, S. Debernardis, and G. Monno, "Augmented reality on large screen for interactive maintenance instructions," Computers in Industry, vol. 65, no. 2, pp. 270–278, 2014.

[6] C. Kamphuis, E. Barsom, M. Schijven, and N. Christoph, "Augmented reality in medical education?" Perspectives on medical education, vol. 3, no. 4, pp. 300–311, 2014.

[7] J. Herron, "Augmented reality in medical education and training," Journal of Electronic Resources in Medical Libraries, vol. 13, no. 2, pp. 51–55, 2016.

[8] E. Z. Barsom, M. Graafland, and M. P. Schijven, "Systematic review on the effectiveness of augmented reality applications in medical training," Surgical endoscopy, vol. 30, no. 10, pp. 4174–4183, 2016.

[9] A. Cirulis and K. B. Brigmanis, "3d outdoor augmented reality for architecture and urban planning," Procedia Computer Science, vol. 25, pp. 71–79, 2013.

[10] X. Wang, "Augmented reality in architecture and design: potentials and challenges for application," International Journal of Architectural Computing, vol. 7, no. 2, pp. 309–326, 2009.

[11] M. Billinghurst, "Augmented reality in education," New horizons for learning, vol. 12, no. 5, pp. 1–5, 2002.

[12] J. Fründ, J. Gausemeier, C. Matysczok, and R. Radkowski, "Using augmented reality technology to support the automobile development," in International Conference on Computer Supported Cooperative Work in Design. Springer, 2004, pp. 289–298.

[13] V. Ng-Thow-Hing, K. Bark, L. Beckwith, C. Tran, R. Bhandari, and S. Sridhar, "User-centered perspectives for automotive augmented reality," in IEEE International Symposium on Mixed and Augmented Reality, 2013, pp. 13–22.

[14] S. Gupta and B. Lohani, "Augmented reality system using lidar point cloud data for displaying dimensional information of objects on mobile phones," ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol. 2, no. 5, p. 153, 2014.

[15] C. K. Lau, C. F. R. Chui, and N. Au, "Examination of the adoption of augmented reality: a vam approach," Asia Pacific Journal of Tourism Research, vol. 24, no. 10, pp. 1005–1020, 2019.

[16] I. Rabbi and S. Ullah, "A survey on augmented reality challenges and tracking," Acta graphica: znanstveni časopis za tiskarstvo i grafičke komunikacije, vol. 24, no. 1-2, pp. 29–46, 2013.

[17] S. L. Kim, H. J. Suk, J. H. Kang, J. M. Jung, T. H. Laine, and J. Westlin, "Using unity 3d to facilitate mobile augmented reality game development," in 2014 IEEE World Forum on Internet of Things (WF-IoT). IEEE, 2014, pp. 21–26.

[18] A. Bąk and M. Wojciechowska, "Using the game engine in the animation production process," in Asian Conference on Intelligent Information and Database Systems. Springer, 2019, pp. 209–220.

[19] A. Juliani, V.-P. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents," arXiv preprint arXiv:1809.02627, 2018.

[20] L. Ramos, "Real-time 3d archviz," 2015.

[21] Wikipedia contributors, "Entity component system — Wikipedia, the free encyclopedia," 2020, [Online; accessed 19-December-2020]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Entity_component_system&oldid=992905855