Avrupa Bilim ve Teknoloji Dergisi
Özel Sayı, S. 244-257, Ağustos 2020

Araştırma Makalesi

www.ejosat.com ISSN:2148-2683

European Journal of Science and Technology
Special Issue, pp. 244-257, August 2020

Research Article

ARgent: A Web Based Augmented Reality Framework for Dynamic Content Generation

Gökhan Kurt[1], Gökhan İnce[2*]

[1] İstanbul Teknik Üniversitesi, Fen Bilimleri Enstitüsü, Oyun ve Etkileşim Teknolojileri Bölümü, İstanbul, Türkiye (ORCID: 0000-0002-8189-6552)
[2] İstanbul Teknik Üniversitesi, Bilgisayar ve Bilişim Fakültesi, Bilgisayar Mühendisliği Bölümü, İstanbul, Türkiye (ORCID: 0000-0002-0034-030X)

(Bu yayın 26-27 Haziran 2020 tarihinde HORA-2020 kongresinde sözlü olarak sunulmuştur.)

(**DOI:** 10.31590/ejosat.779946)

ATIF/REFERENCE: Kurt, G. & İnce, G. (2020). ARgent: A Web Based Augmented Reality Framework for Dynamic Content Generation. *Avrupa Bilim ve Teknoloji Dergisi*, (Special Issue), 244-257.

Abstract
In the modern world, people are more and more interested in interactive technologies. Despite the usefulness of augmented reality, which is a novel addition to those interactive technologies, its development still requires knowledge and experience with programming and game development tools gained by long-term education and training. People experienced in design and content creation can be deprived of the ability to create and maintain AR applications. In this paper, *ARgent*, an AR authoring tool developed in Unity will be introduced. This framework will supply the whole workflow pipeline which involves modules targeting the tree fronts: the server, the web interface and mobile application. The server will be responsible for packaging and optimizing objects, doing database processes and serving the data to the web interface and mobile application. The web interface will be used for content management as web applications are easy to use and easily accessible. The mobile application is the application that will be used by the end user. *ARgent* will provide a way to create applications entirely in a Graphical User Interface (GUI) without needing to write any line of code. The User eXperience (UX) will be familiar to the existing tools to keep users engaged with the development process. *ARgent* provides a way to import dynamic assets, create animations and optionally create scripts for custom behavior, all within its web interface. *ARgent* features "asset bundling" method, a novel way to optimize and load dynamic assets, which eliminates performance issues arising from using dynamic assets in applications built with Unity. Finally, a JavaScript engine is added to the framework so that users can write their own scripts enabling them to create custom user interfaces and to assign behaviors to them.

Keywords: Augmented reality, framework, authoring tool, human computer interfaces

ARgent: Web Tabanlı Dinamik İçerik Destekli Artırılmış Gerçeklik Geliştirme Altyapısı

Öz
Günümüzün dünyasında insanlar interaktif teknolojilere daha fazla ilgi duyuyor. İnteraktif teknolojilerin en yenilikçi örneklerinden biri olan artırılmış Artırılmış gerçeklik (AG), her ne kadar işe yarasa da, AG uygulamaları geliştirmek programlama ve oyun geliştirme araçları hakkında uzun süren eğitimler sonucu kazanılan bilgi ve tecrübeler gerektiriyor. Diğer yandan tasarım ve içerik oluşturma konusunda yetenekli insanlar, AG uygulaması geliştirmekten ve yönetmekten mahrum kalabiliyor. Bu makalede *ARgent*, Unity'de geliştirilen bir AR geliştirme aracı tanıtılacaktır. Bu altyapı iş akışının tamamını kapsayan bir yol haritası temin eder ve sunucu, web arayüzü ve mobil uygulama gibi üç ayrı cephede hizmet verir. Sunucu objelerin paketlenmesi ve optimizasyonu, veritabanı yönetimi ve verinin web arayüzü ve mobil arayüze sağlanması gibi görevleri üstlenir. Web arayüzü kullanımı kolay ve herkes tarafından erişilebilir olması neticesiyle bir içerik yönetim sistemi olarak görev görür. Mobil uygulama ise son kullanıcı tarafından kullanılacak uygulamadır. *ARgent* sadece kullanıcı arayüzünü (GUI) kullanarak ve hiçbir programla yapmaya gerek kalmadan uygulamalar geliştirilmesini sağlayacaktır. Kullanıcılara alışık oldukları bir deneyim sunmak amacıyla, işlemler halihazırda varolan diğer araçlara benzer olacaktır. Unity'de görsellerin dinamik olarak içeri aktarılması sebebiyle ortaya çıkan performans problemleri, *ARgent*'in sunduğu ve yenilikçi bir yöntem olan "asset bundling" yöntemi ile çözülmüştür. Son olarak, altyapı bir

[*] Sorumlu Yazar: İstanbul Teknik Üniversitesi, Bilgisayar ve Bilişim Fakültesi, Bilgisayar Mühendisliği Bölümü, İstanbul, Türkiye (ORCID: 0000-0002-0034-030X), gokhan.ince@itu.edu.tr

Javascript motoru barındırır ve böylece kullanıcılar kendi betiklerini entegre ederek isteğe göre kullanıcı arayüzü ve nesne davranışı tanılmayabilir.

Anahtar Kelimeler: Artırılmış gerçeklik, geliştirme altyapısı, yazarlık aracı, insan bilgisayar arayüzleri

1. Introduction

With the recent developments in interactive technologies, it is now possible merging the virtual media with the real world to create an alternative rendering of reality, called as Augmented Reality (AR). This is an innovative technology that can be utilized in gaming, simulation and enterprise applications. Users immerse more deeply and become more engaged with activities which are reinforced with AR [1]. As a result, many traditional activities are being adapted to AR. AR has lately proven to be an efficient tool in education, training, industry, healthcare, tourism and entertainment applications [2].

AR is an interactive experience that aims to enhance the objects in the real world in real time by using feedback in the form of vision, audio, tactile along with other forms [3]. The AR mentioned in the context of this article is mobile augmented reality, which predominantly is about enhancing reality by putting three dimensional (3D) objects on top of real world imagery and is made available to consumers mostly via their personal smartphones. However, some head-mounted mixed reality (XR) devices can be also considered in the scope of this article. These devices provide extra features such as hand tracking and voice commands, but they follow same principles as AR-capable smartphones.

The AR technology is made possible by highly accurate calculations and computations in the field of computer vision and artificial intelligence. It works by sensing the real environment and deducing the viewer's position and movement, before displaying virtual objects in the environment relative to the viewer [4]. The viewer's position, orientation and motion, as well as the 3D structure of the environment is calculated by using the inertial sensors and running computer vision algorithms on the camera imagery [5]. This is possible thanks to the long years of research and development by large teams and their low-level computing and programming know-how.

It is not feasible for small teams and independent developers to create AR applications by investing so much time and knowledge. That is why there are tools like Unity, Vuforia, ARKit and ARCore that provide ways to develop AR applications without needing that much investment [6]. These tools and frameworks provide Software Development Kits (SDK) and Application Programming Interfaces (API) that abstract away the low-level details. People with reasonable programming experience can create AR applications using these APIs.

However, despite all these advancements, AR technology is not widely adopted. The need for AR applications are usually custom-made for the client's specific needs and to build an AR application requires technical skills in multiple disciplines such as programming, designing, modeling, animating and texturing [7]. To transition AR into becoming mainstream, authoring tools that allow non-programmers to create their own AR applications is crucial [8]. By enabling creating of AR applications by domain experts and designers, who are not technically skilled as programmers but have better understanding of the application's specific needs, the speed of creating will increase and costs will decrease [9].

Unity game engine is the leading tool used in the majority of AR applications. In fact, 60% of AR applications are developed in Unity [10]. Although the cost of finding people capable of developing applications in Unity is high, it is the tool of choice when developing an AR application. Some extensions can be installed over Unity to improve the AR development experience. However, existing tools are hard to use without significant experience and technical expertise.

In this article, to make the development process smoother and easier for non-technical people, *ARgent*, an AR GENeration Tool is proposed. *ARgent* will provide a way to create applications entirely in a Graphical User Interface (GUI) without needing to write any line of code. The User eXperience (UX) will be familiar to the existing tools to keep users engaged with the development process. *ARgent* provides a way to import dynamic assets, create animations and optionally create scripts for custom behavior, all within its web interface. *ARgent* features "asset bundling" method, a novel way to optimize and load dynamic assets, which eliminates performance issues arising from using dynamic assets in applications built with Unity. To demonstrate the applicability of the proposed system, a proof of concept application in marketing will be created using *ARgent*. The simplicity *ARgent* provides and challenges faced with it during the creation of this experimental application will be discussed. Asset loading speed and other performance metrics will also be investigated and discussed.

2. Existing Work

Software abstraction is the purposeful suppression, or hiding, of some details of a process or artifact, in order to bring out other aspects, details, or structure more clearly [11]. Every software framework, language or application is an abstraction that simplifies details of the underlying system. Essentially, the options for creating AR applications are all abstractions hiding the complexity of AR

computations. The frameworks for creating AR applications can be investigated in 3 groups. Those are Software Development Kits (SDK), game engines, and authoring tools. SDKs have lowest level abstraction but are more generalized, while authoring tools have highest level abstraction and are more specialized.

2.1. Augmented Reality Software Development Kits

Augmented reality SDKs are specialized APIs that abstract away the low-level details of computations. Hardware developers publish these SDKs to provide other people ways to easily create AR applications on their hardware. The SDKs work by gathering the data generated by multiple sensors of the device such as gyroscope, accelerometer and magnetometer, and combine this data with the visual imagery gathered by device camera to estimate the state of the real world to aid in the experience. The data is often processed with machine learning algorithms to generate an understanding and interpretation of the real world as well as the position and the orientation of the viewer. Other sensors like Global Positioning System (GPS), depth camera or InfraRed (IR) cameras can be used if available.

Although Software Development Kit (SDK) is a general term for a collection of software tools for making creation of applications easier, in this article, this term will be used specifically when referring to Augmented Reality SDKs. Some capabilities are essential for a good SDK: 1) Motion tracking, which is the process of determining the viewer's position relative to the world, 2) Environmental understanding, which is detecting and understanding the 3D structure of nearby objects and environment like ground, wall, tables, chairs etc. and 3) Light estimation, which aims at estimating the environment's current lighting conditions like the direction and intensity of nearby light sources. The most widely used AR SDKs will be presented below.

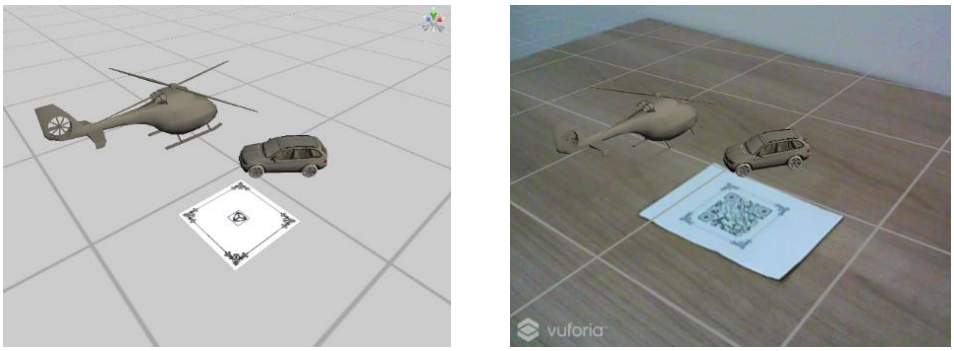
ARKit: ARKit [12] is the AR development framework for Apple devices such as iPhone and iPad. It takes advantage of software and hardware capabilities of these devices to provide features such as motion tracking, environmental understanding, light estimation, multiple face tracking, people occlusion, and motion capture.

ARCore: ARCore [13] is the AR framework for Android devices. It is developed by Google and supports only a subset of Android devices which are mostly manufactured by Google subsidiaries. Its features include, motion tracking, environmental understanding, light estimation.

AR.js: AR.js [14] is a marker-based AR library. The importance of this library is that it runs completely on web, meaning that it is possible to use an AR application built in AR.js just by opening a website link, without installing any application. This is possible by using WebGL, WebXR and WebRTC features of modern web browsers. It is built with JavaScript, three.js and ARToolkit. The main advantages of AR.js are its compatibility across browsers and devices with WebGL and WebRTC features, showing high performance even in old smartphones, no need to install any app as it is web-based, being open source and free, working on a smartphone without additional hardware, and being easy to use and get started.

Vuforia: Vuforia [6] is a cross-platform, target-based SDK. It supports many platforms such as Windows, Android, IOS, and HoloLens running on devices such as AR/XR headsets, smartphones and tablets. A target-based SDK works by analyzing the image acquired by the device camera with a technique called photogrammetry, and searching for pre-determined target to determine the position and orientation of the viewer relative to the target [6]. By having a predetermined target in the real world, the tracking can work more precisely and risk-free, as environmental conditions and inconsistent ambient lighting may harm feature detection [15]. The target can be a planar 2D image or a 3D object. Vuforia supports various types of targets to track model targets, image targets, object targets, cylinder targets, Vumarks, ground plane.

The proposed framework, *ARgent*, utilizes Vuforia features, specifically the image targets, also known as markers. Markers provide the most accurate and stable way to display AR content [16]. In marker-based AR applications, the virtual image is placed on top of the marker when the device camera detects a predefined marker. An example marker-based AR application can be demonstrated as in Figure 1.



(a) Scene view in Unity *(b) AR view*

Figure 1: A marker-based AR application

Vuforia is chosen as the SDK to develop *ARgent* on because of its cross-platform compatibility and phenomenal positional tracking capability. With Vuforia, majority of AR devices including Android, iOS and HoloLens can be supported with a single implementation. This eliminates the need to use SDKs that are targeted to their specific device such as ARCore and ARKit. Although Vuforia does not have other quality improving features such as light estimation, its marker-based motion tracking and environmental understanding is satisfactory for the purposes of *ARgent*. Another candidate to create *ARgent* in was AR.js, which has features compatible with the features planned for *ARgent*. It is cross-device compatible as it is built on web technologies, and web technologies are available in most platforms. However, motion tracking and environmental understanding of AR.js is not as stable as Vuforia's. Vuforia uses various capabilities of the device on the operating system level, whereas AR.js can only access what Web API offers [17].

2.2. Game Engines

Game engines are general purpose tools that exist to help people create video games. Some game engines offer the ability to create AR applications by incorporating the SDKs which were discussed in the previous section. They abstract the details of SDK and provide a unified development environment that incorporates the SDKs for supported hardware.

Unreal Engine: Unreal Engine is a popular game engine. It is in a competitive relationship with Unity and it also has a large user-base. It provides AR capabilities in its latest versions. It allows developing AR applications to Android and IOS platforms using their respective SDKs, ARCore and ARKit.

Unity: Unity is the game engine of choice for most small-sized companies. It is also the leading platform for AR game and application development. Most AR SDKs provide integration to Unity and because of that, there are a lot of options when developing an AR application in Unity. Users can either choose Vuforia if they desire target-based tracking, or ARCore and ARKit if they desire flexibility and fine-tuning. There is also an option to use ARFoundation, which unifies ARCore and ARKit development to make them easily developed from a single API. There are dozens of other AR frameworks which can be integrated into Unity, which shows how much flexibility it offers.

The proposed framework, *ARgent*, is built with Unity and Vuforia plugin. It incorporates the processes that are familiar to the users of Unity in order to preserve the user experience.

2.3. Authoring Tools

AR authoring tools are specialized tools to create AR applications in a specific field or use case. They are niche tools that can improve the authoring process if used in the right context. This kind of tools can be used to rapidly prototype an idea without investing as much time as other alternatives. These tools usually offer a way to abstract away the technical details like programming and allow non-technical people like designers and domain experts to create AR applications. To compensate for the lack of programming options, they may provide visual scripting tools and other GUI based solutions to implement complex logic. The program that is created in the scope of this article, *ARgent*, is also in this category.

3. ARgent Framework

Creating software is a hard task requiring many days of training and research. That is especially true when it comes to games and game related applications, such as AR applications. The reason for this is because developing such applications is multidisciplinary and time consuming, and it is hard to find talented work force for the task. As a result of these, developing AR applications is expensive. It is often not possible to develop large-scale applications without a skilled team. To minimize these problems, and describe a well-defined way to create AR applications, would make the process simpler and cheaper.

Some processes in application development is common between all projects. These processes can be standardized, so that the whole workflow, in other words the pipeline, is well-defined and understandable. In Section 3.1, the overview of the workflow, which constitutes the pipeline of the proposed framework will be explained, whereas Section 3.2 focuses on the implementation aspects of this framework.

3.1. System Overview

The workflow consists of these steps: Uploading assets, importing assets, creating animations, scripting and AR-based application. This workflow can be seen as visualized in Figure 2. This workflow is defined for adding an object to the AR scene. For each object to be added, these steps are repeated. After all the objects are created, the scene is ready to test or deploy.

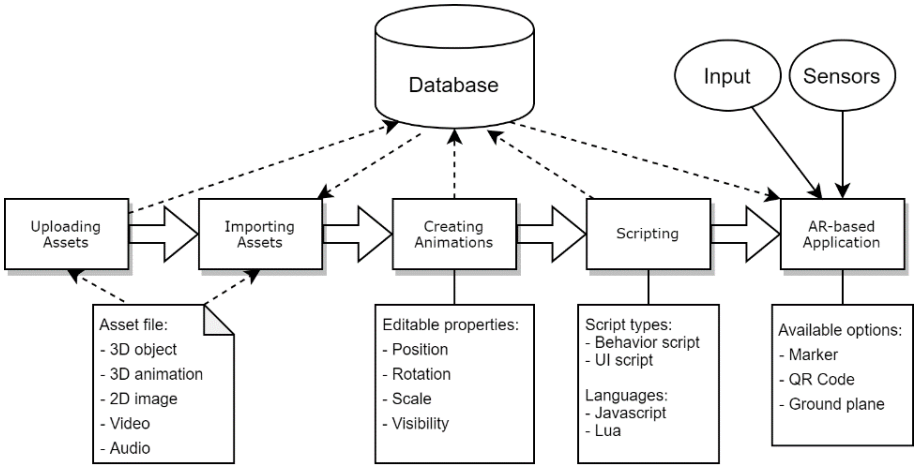


Figure 2: The workflow of the proposed ARgent framework

Each step can be described as follows:

Uploading Assets: This step is for adding an asset to the system. In the traditional application development, this is like adding an asset to a project's file system. In *ARgent*, uploaded assets are processed and saved to the server's database. An asset file can be 3D object, 3D animation, 2D image, video and audio. These are the most used asset types in game development. However, more asset types can be added if required.

Importing Assets: In this step, an asset is imported into the scene. In traditional development, this is analogous to adding an object with an asset as its visual to the scene. The asset must first be uploaded and processed through the ***Uploading assets*** step to be able to import. However, the user interface may provide users an ability to directly import assets, making the ***Uploading assets*** step more transparent, thus improving the overall experience. In traditional development, assets are retrieved from the file system. In *ARgent*, imported assets are retrieved from the server's database, which were already saved to the database in the ***Uploading assets*** step.

Creating Animations: In this step, the designer defines the animation of an object. Animation of an object is defined as the transformation of the object over time. An object's transformation is defined by its properties: position, rotation, scale and visibility. The designer may not want to define an animation for the object. However, the initial transformation of the object is necessary. This can be done by defining the transformation only for the initial time.

Scripting: An application reacts to the changes in the environment and inputs from the user. Also the application may have a custom behavior rather than the simple and repetitive animations. This step allows designers to define behaviors for objects in a scripting language of their choice. This is an optional step, which means designers may skip this step if they do not desire its functionality. A script can be one of the two types: 1) a behavior or 2) a UI script.

Behavior scripts define the behavior of an object. This means that the input from various sources may have an effect on the object. For example, a designer might want to make an object draggable, meaning that a user can move the object by touching on the screen and dragging their finger on a different point. A UI script, however, creates the interface for the application, and the behavior of the interface. For example, designer might want to add a button to the top-right corner of the screen and terminate the application when a user clicks it.

Any scripting language can be used as long as the application's game engine supports the language. *ARgent* is built in Unity and it has the capability of supporting JavaScript and Lua scripting languages. JavaScript is one of the most popular programming languages, hence why it was chosen for the initial development of *ARgent*.

AR-based Application: This step is a testing process rather than an authoring process. In this step, the designer experiences the scene he/she created in an AR environment. The application can be experienced using AR methods such as marker tracking, AR code tracking or ground plane tracking. These methods will be described thoroughly in Section 3.2.1.1. The application receives input from user such as touch input or hand tracking, and from sensors such as the video camera, GPS and inertial measurement unit (IMU) to further improve the experience.

3.2. Implementation of *ARgent* Framework

An application created using *ARgent* is a mobile application capable of viewing dynamic scenes in AR. As discussed in Section 2.1, image targets (markers) are an effective way to track the environment in AR applications. *ARgent* mobile application uses a QR code wrapped by a marker to show the scene on. As the content of the application is going to be dynamic, the content files cannot be included in the build files of an application. The application must allow dynamically uploading and downloading of content. For such an AR application, a generic method to generate content is required, which is what the authoring tool will achieve. It must be kept in mind that the people who are working on content generation are generally not technically competent. The content generation interface should be usable with a low learning curve to these people. Also it should have an intuitive design and UI.

One of the challenges of such an application is performance. The ability to dynamically add new content comes with the drawback of poor performance when importing the dynamic content. In addition to presenting the implementation of *ARgent*, this section will present ways to overcome performance issues, their advantages and disadvantages.

3.2.1. Architecture of ARgent Framework

The content is structured in a way to maximize reusability and group related entities together to have a clear architecture. In order to describe the developed methods clearly, the terminology will be explained using Figure 3:

- A **scene** is a list of objects arranged to form a meaningful composition.
- An **object** is the atomic element shown in AR view. Every object is associated with an asset, and an animation.
- An **animation** is the transformation of an object over time. Accordingly, the position, rotation and the scale of an object can change in a scene.
- An **asset** is the visual representation of an object. An asset can be one of the following types: 3D Model, 3D Animation, 2D Image, Animated Image (GIF), Video, and Audio.

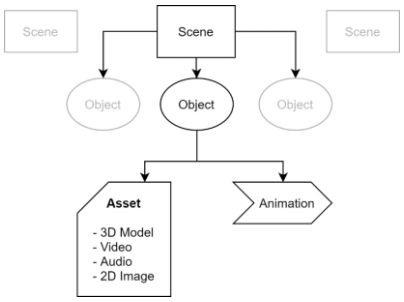


Figure 3: Content structure used in ARgent

From an implementation point of view, the application can be separated into three main parts. The mobile application, the server and the web interface (Figure 4).

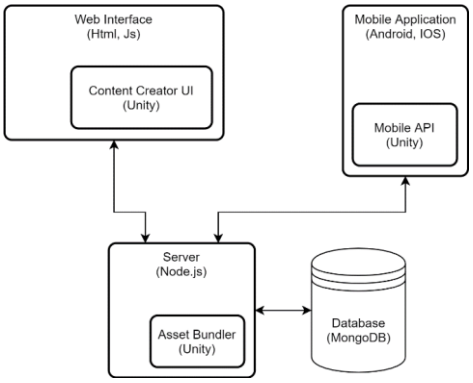


Figure 4: Architecture of ARgent Framework

3.2.1.1. Mobile Application

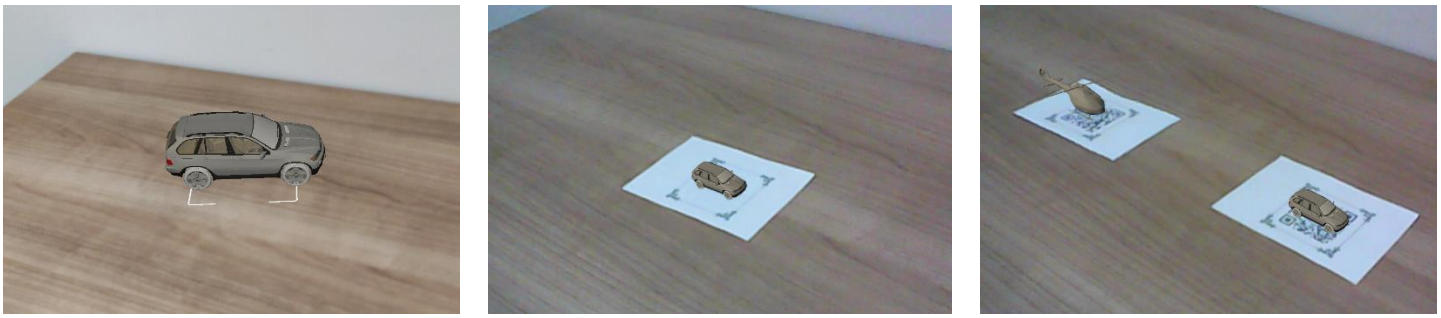
The generated mobile application is the medium, which the end user interacts with. It consists of two parts, wrapper and mobile library.

Wrapper can be thought as an application shell that can include software paradigms as UI, application settings, server calls or other procedures usual mobile applications have. It can be changed in any way to suit the needs of the customers and end-users. Wrapper calls the mobile library at an appropriate time to show the AR content.

Mobile library is the part of the application that shows the AR scene. It is a piece of software containing *ARgent* capabilities. It downloads scenes and models from *ARgent* server and displays it in AR. The mobile library is built in a way to be extensible and easy to integrate into other applications. The library can be included in any mobile application to add AR capabilities to the application, i.e. by wrapping it. The developers of the wrapper can call a predefined function in library with a scene identifier to spawn that scene, or let the mobile library detect scenes to show by scanning AR-code.

In the AR view as exemplified in Figure 1(b), the scene will be shown taking a marker image as reference. This marker image can be a default image or an image provided by the user via the web interface. As seen in Figure 5, the mobile library can typically run in 3 different modes, which all cover a different use case based on the decision of the wrapper. These modes are:

Single scene ground mode: Similar to the single scene marker mode, the wrapper must specify a single scene identifier. The difference of this mode is, the scene will be shown when the mobile library detects a ground (Figure 5(a)). This mode is more convenient in use cases where the scene must be shown on a flat floor. This mode can be used to quickly view scenes in any place since a marker is not needed to be printed out. The mobile library uses ARCore's ground plane tracking feature to make this mode work.



(a) Ground mode *(b) Single marker* *(c) Multiple markers with QR inside them*

Figure 5: Different rendering modes

Single scene marker mode: In this mode, the wrapper must specify a single scene identifier along with other mode parameters. The mobile library will show this scene as soon as it detects the marker (Figure 5(b)). The standard marker image must be printed and placed/mounted/hanged in the real world. The mobile library uses Vuforia's image tracking feature to make this mode work.

QR mode: In this mode, the wrapper does not have to specify a scene identifier. The mobile library needs to detect a marker and a QR code near the marker for this method to work. When a QR code is detected near a marker, it is treated as a scene identifier and the scene is superimposed on top of the marker (Figure 5(c)).

One of the contributions of this article is to introduce the combination of a QR code and an AR marker, which can be called an AR-code. For this method, every scene has a different AR-code which contains the QR coded identifier and the AR marker which the *ARgent* mobile application can recognize. The AR-code can be generated in two ways (Figure 6). First approach relies on the AR marker framing the QR code (Figure 6(b)). Second way is called a branded QR code, where QR code frames the AR marker [18] (Figure 6 (c)).

Both ways can work well most of the time. The second way may look better from a design perspective; however, in some cases the AR marker can be very small for the application to accurately track the marker position [19]. Thus, the second way is not always preferable.

(a) Original marker *(b) QR code wrapped by marker* *(c) Marker wrapped by QR code*

Figure 6: Difference of two ways of generating AR-codes

3.2.1.2. Web Interface

The web interface is the medium through which content creators interact with the application. Web interface provides content creators a way to generate content for the AR application. Using this interface, they can create scenes, put objects in the scene, animate objects and select the asset associated with the object.

Adding objects: When adding objects, user will be asked to provide an asset for the object. This asset can be uploaded instantaneously or one of pre-defined assets uploaded before, can be selected. Unlike uploading a new asset, a pre-defined asset will be immediately loaded and it is more efficient to use this option instead of uploading a duplicate of the asset. The reason for this is explained in Section 3.2.2. The web interface has a page called "Asset Manager" where users can manage the pre-defined assets, upload a new version for an asset, and delete assets. It can be seen in Figure 7.

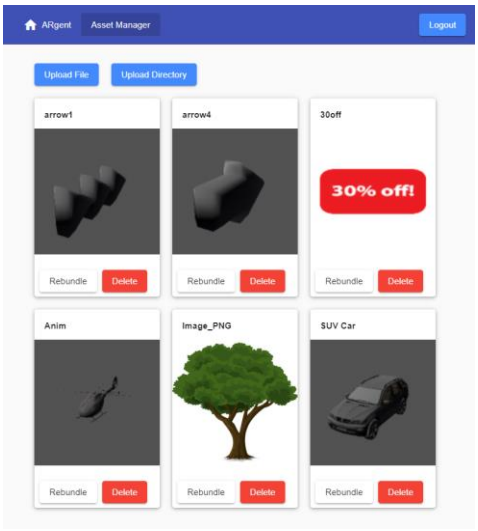


Figure 7: Asset manager page

The web interface has a comprehensive UI element called the hierarchy panel (Figure 8). At the top of the panel ①are the tools used to modify the properties of the object. There are buttons for *undo*, *redo*, *translate*, *rotate* and *scale*. There is also a set of inputs where values for the transform of the object to be put in manually ②or by copy and paste ③.Below the panel the object hierarchy can be seen ④.Objects can be added by clicking the add button ⑤, which will open the asset select interface. Objects can be selected by clicking on their name. After selecting, the properties of the object can be edited, or the object can be deleted ⑥.Objects can also be selected by clicking them in the 3D view.

Defining animations: Users can define position, rotation and scale of an object at different points in time to create an animation. Objects can also be hidden and shown as part of animation. The process of creating animations is carried out in similar professional 3D animation software, but it is also easy to use for newcomers and non-technical people.

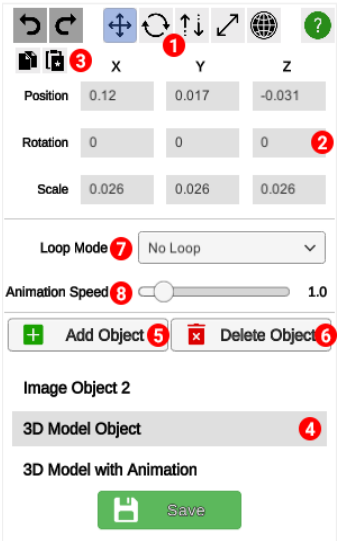


Figure 8: Hierarchy panel in the web interface of ARgent Framework

To start creating an animation, users must first select an object by clicking them in the hierarchy panel, or the 3D view as in the left panel of Figure 3.8. After that, when changing the transformational properties of an object, the properties will be saved for the current time frame. A transformational record saved for a time frame is called a keyframe. To select a different time frame, users can use the timeline at the top of the screen. Saved keyframes will also be shown in the timeline, where they can be modified or deleted.

Users can select the looping method of animation (Figure 8 ⑦). An animation can be defined to not loop, loop infinitely or do a loop alternating between forward and backward animation, also called ping-pong loop. Users can also set the animation speed on the hierarchy panel (Figure 8 ⑧). The transform of an object in the time between two keyframes will be calculated by interpolation, which is called tweening. The easing function of interpolation, which affects smoothness of the animation, can be defined by the user. Clicking on the green bar between two keyframes (Figure 9) will switch between easing functions for that interval and the bar will change color accordingly. Some easing functions like linear, cubic and sine interpolation are supported within *ARgent* Framework.

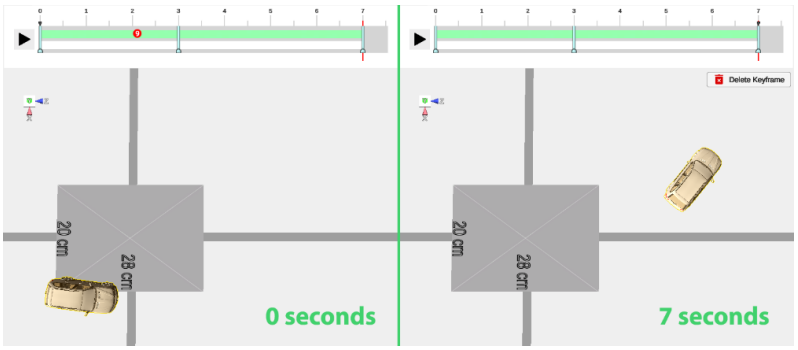


Figure 9: Animation workflow in the web interface

3.2.1.3. Server

Responsibilities of the server can be summarized as follows:

Providing a REST API for the web interface and the mobile application: The REST API provides endpoints of create, read, update and delete (CRUD) operations for assets, scenes, scene objects and animations. It also provides endpoints for uploading assets, downloading assetbundles, preview images and original asset files. The API is written using *node.js* and *express.js*.

Persisting the data in the database: The data modified by the CRUD operations are saved in the database. Assets, scenes, scene objects and object animations all have metadata that need to be kept in the database. The database uses *MongoDB* technology, which is useful with the polymorphic and data oriented approach this application uses.

Handling and processing the uploaded assets: Assets uploaded through the web interface will be processed by the server in order to be used by the web application or the mobile application. Users are able to upload assets like 3D objects, 3D animations, images, animated images, videos and audio clips. As the application is implemented in Unity3D, there is no good way to make the raw files usable in runtime. The uploaded raw files need to be processed priorly to be able to use them in the mobile application or the

web interface. To overcome this issue, two novel implementations are introduced in the *ARgent* Framework. These implementations will be explained in Section 3.2.2.

Keeping original and processed assets in the file system: The server groups all the files related to an asset in a single folder, named with the asset identifier. The folder contains the original asset file, processed asset file, high definition preview image and preview image thumbnail. If the uploaded asset was a zip file originally, both the original zip file and the extracted directory will be stored. If the uploaded asset was a directory, the whole directory will be stored. Only one version of an asset is stored, meaning that if the asset is uploaded again, old asset files are overwritten. This is done in order to reduce impact on the disk space.

3.2.2. Improvements on the Server Implementation of ARgent

When an asset is uploaded to *ARgent* server, the asset must be processed in order to be usable in AR applications. There are two ways to process assets dynamically. These two methods are dubbed as "Runtime Parsing" which is the existing approach and "Asset Bundling" which is the novel method having significant advantages over the first method.

3.2.2.1. Runtime Parsing

In this method, assets uploaded by the server are not processed by the server. Instead original files are kept in the file system. When the client requests the asset, original file is sent and client must process it by parsing it in the runtime. It is trivial to parse most basic file formats like `.obj' for 3D models and `.bmp' for images. These files are the simplest file formats for their respective media. They keep the data uncompressed and unencrypted, which makes them easier to parse. However, they are not widely used because their file sizes are rather large compared to the compressed alternatives. In fact, size of .bmp files are very big and they are not suitable for uploading, downloading and efficiently storing images in file system [20]. They should only be used for archiving and high quality printing purposes. As for the `.obj' files, they do not support the critical features that other 3D model formats have.

Parsing becomes harder when the file formats are complicated as different encryption, compression and data representation mechanisms are taken into consideration. Even the most common 3D model file formats like .fbx and .3ds need considerable computational power to convert them into usable data. Also every file format has a different format specification and a special parsing mechanism must be implemented for each file format.

Another disadvantage of this method is that it puts significantly more computational load on the client side. This means that only the high-end devices will be able to use the mobile application. Also, 3D applications running in web browsers are CPU consuming even without this parsing method. With this parsing method, they become much slower and less responsive. Fortunately, Unity supports runtime parsing of .png and .jpg files. This makes it advantageous to use runtime parsing for these files. For other files, however, the proposed asset bundling method will be used within the *ARgent* Framework.

3.2.2.2. Asset Bundling

Unity provides a way to store some content of the application into different files other than the application itself. These files are called assetbundles. Assetbundles can be downloaded by the application at any time to provide content to the application dynamically. They are usually uploaded into remote servers and fetched in a later date to either reduce the application's file size, or to provide content to the application regularly. However, creating assetbundles is a tedious task. They can only be created in the Unity Editor and they require a fair amount of programming skill to implement. This requires technical skill on the content creator's side to do it manually.

ARgent's asset bundling method uses Unity's built-in asset bundling mechanism as its name implies. The server converts assets into assetbundles right away when they are uploaded. However, instead of doing it manually, the server automates the process of creating assetbundles. The automation of asset bundling is accomplished by copying the asset files into a Unity project that was prepared beforehand. Then, a Unity Editor process is spawned and starts running the project. Consequently, the server tells Unity Editor process to create an assetbundle. Upload process is completed after generated assetbundles are copied back to the server's database.

A different assetbundle is created for each platform. This is because the Unity Runtime handles assetbundles differently in different platforms. This application runs on platforms such as WebGL, Windows, Android and iOS. As a result, four assetbundle files are generated for these four platforms. If there is a need to support more platforms in the future, they can be supported as long as Unity supports creating assetbundles for those platforms. Because the parsing and processing is done by the Unity Editor on the server side, these methods allows all file formats supported by Unity Editor to be used. The parsing process also becomes faster and the parsing only happens once, when the asset is first uploaded.

This method has another advantage that enables users to create a preview image for the uploaded asset. Since the Unity Editor is started up for this method to work, the uploaded asset is put in an empty scene and a screenshot is taken. This screenshot acts as a preview image for the object. The preview image can be used in places like asset manager in the authoring tool to make users be able to quickly identify the asset by looking at it.

4. Experiments and Results

In order to objectively evaluate the benefits and limitations of *ARgent*, a proof of concept application was created. This section will describe this case study, walk the reader through the end-to-end process of creating an application with *ARgent*, and discuss the performance of the proposed asset bundling approach.

4.1. Proof of Concept Application of *ARgent*

In the scope of this article, an experimental AR application, acting as an advertising campaign for a fictional supermarket was created. In this fictional ad campaign, some of the products on the supermarket's shelves have a discount, and customers can find them using the AR application. Every product on the shelves have an AR-code that the customer can scan with the *ARgent* mobile application to show the AR content assigned to the product. Thus, the shopping task becomes a gamified and engaging experience for the customers. To create this application, a scene for each product in the supermarket must be created. By creating the scene and giving it a suitable name, AR content of the scene is ready to modify.

A wholistic view of *ARgent* web interface can be seen in Figure 10. To add assets to the scene, user must click the **Add Object button**, which will give the user an option to import a previously uploaded asset, or upload a new asset from the computer. When the import process is finished, user can select the object from the **hierarchy panel** or by left-clicking on it in the 3D view. Properties like transformation of an object can be edited through **transformation controls** when the object is selected. The **origin point**, **orientation indicator** and **horizontal plane grid** exist to aid the user when navigating the 3D view. User can also create an animation for the object using the **animation panel**, or upload a script asset for custom behavior with the **Add Object button**.

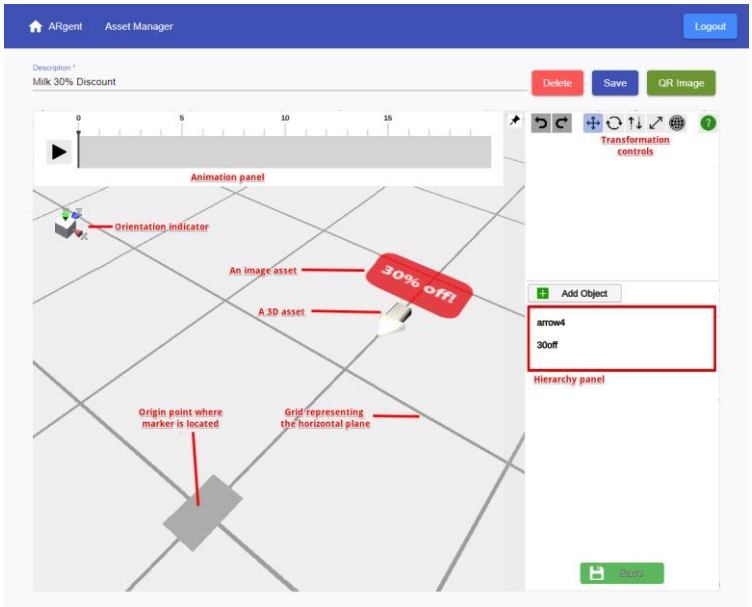


Figure 10: An explanation of the web interface and scene creation process

For the purpose of this application, **an image object** and **a 3D object** is imported and positioned as seen in Figure 10. The 3D object is an arrow pointing to the product (↓), and the image is the text for the discount (30% off!). The space between arrow and the marker is where the product will appear in the AR view.

After editing of the scene's content is finished, the scene is ready to be viewed in the AR application. Clicking the **QR Image button** in the web interface views the AR-code for the scene as in Figure 11 (a). The user must print-out this image and place it next to the associated product on the shelf.

(a) Marker containing scene ID *(b) AR view*

Figure 11: AR content shown relative to the detected marker containing the scene ID

When customers scan the AR-code with the *ARgent* mobile application, they will see the scene being rendered over the product as can be seen in Figure 11 (b). The scene creation process must be repeated for each product. The application can be shipped to customers when a new scene is created for each of the subject products. If the supermarket wants to modify the discount for a product, the scene can be modified via the web interface. Scene can be deleted completely to lift the discount from the product. The benefit of *ARgent* in this context is that the application will not need to be shipped to customers again when a discount is modified, because they will receive the latest changes over the *ARgent* server when they scan an AR-code. Another benefit of *ARgent* is that it provides a simple way to modify the AR content, even a regular non tech-savvy supermarket worker will be able to use it. The web interface does not require installation of any application and is accessible by any computer having a web browser and internet connection.

A limitation of *ARgent* revealed by this experiment is that, a scene must be created for each product individually. Maintaining content may become time consuming if the number of products is high. Currently, *ARgent* does not provide a way to mitigate this problem by automating the process. Although it is possible to create a script to achieve this, it will not be efficient as other tools like Unity may provide a better scripting support.

4.2. Evaluation of Asset Bundling

The two dynamic asset importing methods, runtime parsing and asset bundling, both have their own benefits and drawbacks. The two methods are compared in different aspects such as supported file formats, loading speed, hardware requirements and opportunities for improvement they present.

File Formats: As seen in Table 4.1, the asset bundling method supports drastically more file types. Parsing .jpg and .png files in runtime is already supported by Unity so runtime parsing method will be used for these formats. For all other formats, asset bundling will be used.

Table 4.1: Supported file formats of runtime parsing and asset bundling

Asset Type	Runtime Parsing	Asset Bundling
Image	png, jpg, bmp	png, jpg, bmp, psd, tiff
Model	obj	obj, dae, fbx, 3ds, blend, maya
Video	N/A	mov, mpg, mpeg, mp4, avi, asf
Audio	N/A	mp3, wav, ogg

Loading Speed: Table 4.2 shows that although in some cases asset bundling method may take more time in initial loading; for subsequent loadings asset bundling will always load drastically faster than runtime parsing. A reason why asset bundling takes too much time initially is because it needs to generate assetbundles for every platform, hence loading time is increased by four times. However, after this initial cost is paid, every subsequent loading will be fast.

Table 4.2: Loading speeds of runtime parsing and asset bundling

	Runtime Parsing	Asset Bundling
Initial loading	5-60 seconds	40 seconds
Subsequent loadings	5-60 seconds	**1 second**

Hardware Requirements: One of the concerns is the CPU and memory consumption of these two methods. While these metrics are not measured, the cost of using runtime parsing can be felt as it makes the application significantly unresponsive and slow. This effect is not felt when using the asset bundling method.

One of the downsides of asset bundling method is, it requires a Unity Editor to be installed on the server host. This can be a significant dev-ops issue, because most online or cloud hosting platforms usually do not offer this service. A dedicated server host must be used and they cost significantly more. While the runtime parsing required no additional setup on the server side, asset bundling will require a Unity Editor to be setup and running in the server. This implies some limitations: 1) The server must match all requirements of Unity, 2) The server must have Windows Operating System, 3) The server must have a GPU, 4) The server must have more hard-disk space, 5) Unity build plugins of all supported platforms, Windows, WebGL, Android, and iOS, must be installed, and 6) A license of Unity must be set up.

Considerations for Improvement: Processing asset files on the server opens up other possibilities that are not available in runtime parsing method. Firstly, this makes it possible to upload a compressed .zip file which contains the asset. The .zip file is extracted and the result is treated as a directory. Naturally, uploading full directories are also possible and the process is more or less like .zip file upload except the extracting part.

Another possibility asset bundling method opens is the ability to convert between file formats. Some file formats not supported by Unity can be handled by converting them on the server side to a supported file format. The conversion can be done by a third party library which is normally not integrated to Unity. For example, .flv files that are normally not supported by Unity can be converted to .mpeg using *ffmpeg* library, because .flv is not supported in Unity whereas .mpeg is. In fact, .gif files are not supported by Unity but are handled by this method. The support for .gif files is crucial because .gif is a popular animated image format that is used in social media and other campaigns [21].

Some advanced features are also made possible with the asset bundling method. *ARgent* is developed mainly for ease of use and fast learning. However, more advanced features can still be utilized by users who are interested in more advanced scenarios. Since the asset bundling method is using the Unity Editor, any action that is possible in the Unity Editor is technically possible to perform with it.

One of the advanced feature that can be supported this way is full 3D animations. Currently, basic animations are supported. However, there are more features to support, such as animation blending, weighted animations, Avatar feature, runtime animation generation and animations controlled by a finite state machine. With the asset bundling method it can also be possible to import full Unity 3D Scenes in runtime. For this feature, content creators would create a scene in Unity Editor, and upload it as an asset to this application. The scene then would be assetbundled like other assets and be ready to be used in *ARgent*.

5. Conclusion

With *ARgent*, it is possible to create content for AR applications using a web interface. The content of the application is also dynamic, meaning that it can be modified after the application is already shipped. The modified content is retrieved through the *ARgent* server dynamically and cached in the mobile device to provide faster loading the in subsequent runs. The proof of concept study has shown that *ARgent* provides a dynamic content experience without the cost of hindered performance. This is made possible by using a hybrid approach of two dynamic asset importing methods, asset bundling and runtime parsing. Comparison of the two dynamic asset importing methods shows that the runtime parsing cannot handle most file formats that are common to the 3D application environment, whereas most common 3D model formats like *.fbx* and *.3ds* can only be parsed with asset bundling method. With the asset bundling method, there is a long loading the on the first upload of an asset. However, it is not as critical as the loading time on the mobile application side. Also, it will happen only initially instead of on every viewing of the scene as with the case on the mobile application side.

One disadvantage of asset bundling method is that more processing power is needed on the server side. It also implies other requirements on server side, which can make maintaining the server harder. However, this is a small cost compared to the benefits of using this method and can be neglected. The asset bundling method is more favorable compared to the runtime parsing method and it is indispensable method of handling dynamic asset importing scenarios. The lack of scripting options in *ARgent* has been shown to be

a limitation. Future work on this subject aims to improve the scripting capabilities *ARgent* and add an option to visually program behaviors and UI.

References

[1] Seal, A., (2020), Top 7 Augmented Reality Statistics for 2020 [+ Use Cases], https://www.vxchnge.com/blog/augmented-reality-statistics, date retrieved: 09.06.2020.

[2] Cox, L., (2016), 10 Industries Embracing Augmented Reality, https://disruptionhub.com/industries-embracing-augmented-reality/, date retrieved: 09.06.2020.

[3] Höllerer, T. and Feiner, S. (2004). Mobile augmented reality, Telegeoinformatics: Location-based computing and services, 21.

[4] Van Krevelen, D. and Poelman, R. (2010). A survey of augmented reality technologies, applications and limitations, International journal of virtual reality, 9 (2), 1–20.

[5] Chai, L., Hoff, W.A. and Vincent, T. (2002). Three-dimensional motion and structure estimation using inertial sensors and computer vision for augmented reality, Presence: Teleoperators & Virtual Environments, 11 (5), 474–492.

[6] Linowes, J. and Babilinski, K. (2017). Augmented Reality for Developers: Build practical augmented reality applications with Unity, ARCore, ARKit, and Vuforia, Packt Publishing Ltd.

[7] Ramirez, H., Mendivil, E.G., Flores, P.R. and Gonzalez, M.C. (2013). Authoring Software for Augmented Reality Applications for the Use of Maintenance and Training Process, Procedia Computer Science, 25, 189 – 193, 2013 International Conference on Virtual and Augmented Reality in Education.

[8] Seichter, H., Looser, J. and Billinghurst, M. (2008). ComposAR: An intuitive tool for authoring AR applications, 2008 7th IEEE/ACM International Symposium on Mixed and Augmented Reality, pp.177–178.

[9] Lécuyer, F., Gouranton, V., Reuzeau, A., Gaugne, R. and Arnaldi, B. (2019). Authoring AR Interaction by AR, ICAT-EGVE 2019 – International Conference on Artificial Reality and Telexistence – Eurographics Symposium on Virtual Environments, Tokyo, Japan, pp.1–8.

[10] Gajsek, D., (2020), Unity vs Unreal Engine for XR Development: Which One Is Better, https://circuitstream.com/blog/unity-vs-unreal/, date retrieved: 31.05.2020.

[11] (2019), A Detailed Guide to Abstraction in Software with Examples, https://thevaluable.dev/abstraction-type-software-example/, date retrieved: 02.06.2020.

[12] Bohon, C., (2019), Apple's ARKit: Cheatsheet, https://www.techrepublic.com/article/apples-arkit-everything-the-pros-need-to-know/, date retrieved: 02.06.2020.

[13] Lanham, M. (2018). Learn ARCore - Fundamentals of Google ARCore: Learn to build augmented reality apps for Android, Unity, and the web with Google ARCore 1.0, Packt Publishing.

[14] Egington, K., (2019), AR.js: A guide to developing an augmented reality web app, https://3sidedcube.com/ar-js-a-guide-to-developing-an-augmented-reality-web-app, date retrieved: 07.06.2020.

[15] Kasapakis, V., Gavalas, D. and Dzardanova, E., (2018). Robust Outdoors Marker-Based Augmented Reality Applications: Mitigating the Effect of Lighting Sensitivity, pp.423–431.

[16] Zvejnieks, G., (2019), Marker-based vs markerless augmented reality: pros, cons & examples, https://overlyapp.com/blog/marker-based-vs-markerless-augmented-reality-pros-cons-examples, date retrieved: 07.06.2020.

[17] Hermes, (2019), Web vs App (AR edition), https://medium.com/agora-io/web-vs-app-ar-edition-d9aafe988ba2, date retrieved: 13.06.2020.

[18] Hay, D. (2012). The Bootstrapper's Guide to the Mobile Web: Practical Plans to Get Your Business Mobile in Just a Few Days for Justa Few Bucks, Bootstrapper's Guide, Linden Publishing.

[19] Etienne, J., (2017), AR-Code: a Fast Path to Augmented Reality, https://medium.com/arjs/ar-code-a-fast-path-to-augmented-reality-60e51be3cbdf, date retrieved: 07.06.2020.

[20] Hoffman, B., (2012), Unsuitable Image Formats for Websites, https://zoompf.com/blog/2012/04/unsuitable-image-formats-for-websites, date retrieved: 09.06.2020.

[21] (2018), 5 Reasons to You Should Be Using GIFs in Your Social Media Campaigns, https://www.socialreport.com/insights/article/115005444323-5-Reasons-to-You-Should-Be-Using-GIFs-in-Your-Social-Media-Campaigns, date retrieved: 09.06.2020.