ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE

ARGENT: A WEB BASED AUGMENTED REALITY FRAMEWORK FOR DYNAMIC CONTENT GENERATION

M.Sc. THESIS

Gökhan Kurt

Department of Computer Engineering

Game and Interaction Technologies Programme

JUNE 2020

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE

ARGENT: A WEB BASED AUGMENTED REALITY FRAMEWORK FOR DYNAMIC CONTENT GENERATION

M.Sc. THESIS

Gökhan Kurt
(529171007)

Department of Computer Engineering

Game and Interaction Technologies Programme

Thesis Advisor: Asst. Prof. Dr. Gökhan İNCE

JUNE 2020

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

ARGENT: WEB TABANLI DİNAMİK İÇERİK DESTEKLİ ARTIRILMIŞ GERÇEKLİK GELİŞTİRME ALTYAPISI

YÜKSEK LİSANS TEZİ

Gökhan Kurt
(529171007)

Bilgisayar Mühendisliği Anabilim Dalı

Oyun ve Etkileşim Teknolojileri Programı

Tez Danışmanı: Dr. Öğr. Üyesi Gökhan İNCE

HAZİRAN 2020

Gökhan Kurt, a M.Sc. student of ITU Graduate School of ScienceEngineering and Technology 529171007 successfully defended the thesis entitled **"ARGENT: A WEB BASED AUGMENTED REALITY FRAMEWORK FOR DYNAMIC CONTENT GENERATION"**, which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Asst. Prof. Dr. Gökhan İNCE** Istanbul Technical University	.....
Jury Members :	**Assoc. Prof. Dr. Hatice KÖSE** Istanbul Technical University	.....
	Asst. Prof. Dr. Övgü ÖZTÜRK Bahçeşehir University	.....

Date of Submission : **15 June 2020**
Date of Defense : **17 July 2020**

FOREWORD

This thesis is made as a master project, as part of the requirements for the awarding of a degree in Master of Science in Engineering at the department of Game and Interaction Technologies at the Istanbul Technical University. I wish to thank my committee members who were more than generous with their expertise and precious time. A special thanks to Asst. Prof. Dr. Gökhan İNCE, my advisor for his countless hours of reflecting, reading, encouraging, and most of all patience throughout the entire process. I would like to thank all the participants for their contribution in the user experience tests. Finally, I would thank to my family for supporting me all the time.

JUNE 2020 Gökhan Kurt

TABLE OF CONTENTS

Page

ABBREVIATIONS

2D : 2 Dimensional
3D : 3 Dimensional
API : Application Programming Interface
AR : Augmented Reality
CPU : Central Processing Unit
CRUD : Create, read, update, delete
GPS : Global Positioning System
GPU : Graphics Processing Unit
GUI : Graphical User Interface
ID : Identifier
IMU : Inertial Measurement Unit
IR : Infrared
ITU : Istanbul Technical University
OS : Operating System
R&D : Research and Development
SDK : Software Development Kit
UI : User Interface
UX : User Experience
VR : Virtual Reality
XR : Mixed Reality

LIST OF TABLES

Page

LIST OF FIGURES

Page

ARGENT: A WEB BASED AUGMENTED REALITY FRAMEWORK FOR DYNAMIC CONTENT GENERATION

SUMMARY

In the modern world, people are more and more interested in interactive technologies. Education, research and business habits are effected by this change and humans can be more efficient using interactive technologies. Augmented reality (AR), which is a novel addition to those interactive technologies, is especially effective in this matter. Through augmented reality, people can immerse more deeply with the subject experience and they can have enhanced interaction.

Despite the usefulness of augmented reality, it may not always be efficient develop an AR application in terms of development cost. AR development still requires knowledge and experience with certain tools and frameworks. Such tools are usually programming and game development tools and they require programming and technical skills that is 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.

Nowadays, tools like Unity, Vuforia, ARKit and ARCore provide ways to develop AR applications without the need to have knowledge of low-level calculation and programming that is required for AR technology. Normally, developing an AR application would have taken years of research and development by large teams, but thanks to SDK and APIs provided by these tools, AR applications can be developed by small development teams easily and quickly. However, AR is still not easily accessible by all the tech-savvy people that may be interested in developing such applications.

Majority of AR applications are developed using Unity. There are visual programming solutions in Unity, but they are not suitable to be used in AR applications. A Unity-based tool that allows people without programming skills to create AR applications, will be utmost useful. Such a tool would require features such as, creating an application without programming, optional support to do programming and scripting, real time updates and ability to ship without any build and packaging step, support for 3D object, image and video, the ability to modify objects and preview them in real time, and the ability to create user interfaces. The tool should also have a user friendly interface and experience. It should introduce the innovative features without changing the conventional workflows. Existing tools do not provide these features which are crucial for an ordinary person to create AR applications.

In this thesis, a framework 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. It will be possible to integrate the mobile application's core library to any other mobile application as a library.

An AR application created by this framework consists of scenes. Every scene is comprised of objects and every object is assigned a visual asset. A visual asset can be a 3D object, image or video. Also it is possible to assign animation to every object. An animation is defined by sequence of positions in a timeline. By transitioning the object through these positions as time passes, it gives the impression of an animation.

In the web interface, a scene can be created, objects can be put into scene, and position and animation of an object can be defined. Generated scenes are able to be previewed on the web interface without any need to build and package. Thus, users can test the generated scenes without needing a mobile device or emulator. Also, using the content management system in the web interface, users are able to modify visual assets. New assets can be added and existing assets can be modified or deleted.

In the server side, the data coming from the web interface is processed and stored to be served to the mobile application when needed. Visual assets coming from the web interface undergo a packaging step in order to provide faster loading and rendering speed. Thus, the web interface and mobile application can quickly reuse these assets. Scenes and assets are cached, so that they are served quickly without extra calculation steps if there are no changes on them.

The mobile application is an AR application to be used by the end-user. Just like a normal AR application, it searches for a target to augment. When it finds a target, it augments the display with the scene defined in the web application. The target can be an image marker, QR code, or ground plane. Mobile application gets the scene data from the server when it visualizes the scene first time. Although, it needs internet connection for this, for subsequent renderings, it does not require access to the server, because the scene will be pre-cached already.

There are challenges when developing such a framework. The most prominent challenge is performance issues. Performance issues arise because the visual assets of objects are defined dynamically. Under normal circumstances, the assets are optimized and packaged before serving them to the end-user. However, expecting the user to undergo these optimization steps will hinder the user experience. A remedy for this issue is the novel 'asset bundling' method. It works by using Unity's internal optimization pipeline on visual assets in the server side. Experiments show that this method provides benefits in terms of performance.

Users ocassionally want to do extra programming and scripting on top of existing program. This is another challenge that needs to be addressed. To remedy this, a Javascript engine is added to the framework and users can use this to write their own scripts. This engine can also enable users to create custom user interfaces and assign behaviors to them.

Experiments show that developing this framework can be possible without hindering performance significantly. Changes made by users can be seen in the mobile application without the need of any build or release steps. User experience tests show that the feature to update the content of the mobile application without the need to reinstall is beneficial.

ARGENT: WEB TABANLI DİNAMİK İÇERİK DESTEKLİ ARTIRILMIŞ GERÇEKLİK GELİŞTİRME ALTYAPISI

ÖZET

Günümüzün dünyasında insanlar interaktif teknolojilere daha fazla ilgi duyuyor. İnsanların öğrenme, araştırma ve çalışma gibi davranışları da bu değişimden etkileniyor ve interaktif teknolojiler ile yapıldığında daha verimli hale gelebiliyor. İnteraktif teknolojilerin en yenilikçi örneklerinden biri olan artırılmış gerçeklik teknolojisi bu hususta öne çıkıyor. Artırılmış gerçeklik sayesinde insanlar kendilerini ilgili deneyime daha derin bir şekilde dahil edebiliyorlar ve daha kolay bir şekilde etkileşim gerçekleştirebiliyorlar.

Artırılmış gerçeklik (AG) her ne kadar işe yarasa da, AG uygulamaları geliştirmek masraf bakımından buna değecek kadar kolay olmayabiliyor. AG geliştirmek hala belirli araçları kullanmakta deneyimli olmayı gerektiriyor. Bu tür araçlar genellikle programcılara ve oyun geliştiricilere yönelik oluyor ve uzun eğitimler sonucunda kazanılan programlama ve diğer teknik bilgileri gerektiriyor. Tasarımcılık veya içerik üreticiliği gibi tecrübelere sahip insanlar ise AG uygulamaları geliştirmekten ve gelişimine müdahele etmekten mahrum kalabiliyor.

Günümüzde Unity, Vuforia, ARKit ve ARCore gibi araçlar, AG teknolojisini gerçek kılan düşük seviye hesaplamalar ve programlamaya ihtiyaç duymadan AG uygulamaları geliştirilebilmesini sağlıyor. Normalde ancak uzun yıllar sürecek ve büyük ekipler gerektirecek araştırma ve geliştirme çalışmaları sonucunda üretilebilecek AG uygulamaları, bu araçların sunduğu SDK ve API'ler sayesinde küçük ekipler tarafından da yapılabiliyor. Fakat AG henüz bu tür uygulamaları geliştirmek isteyecek her insan tarafından kolayca ulaşılabilir bir durumda değil.

AG uygulamalarının büyük çoğunluğu Unity kullanılarak geliştiriliyor. Unity, görsel programlama seçenekleri sunsa da, bunlar AG uygulamalarına uygun yöntemler değil. Programlama bilgisi olmayan insanların da geliştirme yapabileceği bir Unity eklentisi bu hususta çok işe yarayacaktır. Böyle bir eklenti, programlama yapmadan uygulama geliştirebilme, isteğe bağlı olarak programlama yapabilme, uygulamanın herhangi bir paketleme aşamasına ihtiyaç duymaması ve gerçek zamanda güncellenmesi, 3 boyutlu obje, resim, video gibi objeleri desteklemesi, objeleri gerçek zamanda değiştirme ve canlı olarak önizleme yapabilme ve uygulamaya arayüz geliştirebilme gibi özelliklere sahip olmalıdır. Aynı zamanda bu uygulama kullanıcı dostu bir arayüz ve deneyim sunmalıdır. Alışılagelmiş iş akışlarının dışına çıkmadan, yeni özellikleri mümkün kılan değişikliklere yer vermelidir.

Bu tezde Unity üzerinde geliştirilen bir altyapı tartışılacaktır. Bu altyapı sunucu, web arayüzü ve mobil uygulama gibi çeşitli platformlar gerektiren iş akışının tamamını karşılayacak nitelikte olacaktır. Sunucu tarafında objelerin paketlenmesi, veritabanı işlemleri ve verinin Web arayüzü ve mobil uygulamaya sunulması gibi işlemler

yapılacaktır. Kullanım kolaylığı ve erişilebilirlik açısından, içerik yönetimi için Web arayüzü kullanılacaktır. Mobil uygulama ise son kullanıcının kullanacağı AG uygulaması olacaktır. Ayrıca oluşturulan mobil uygulama altyapısı başka uygulamaların içine gömülebilecek bir kütüphane olacaktır.

Bu altyapı ile oluşturulan bir AG uygulaması sahnelerden oluşacaktır. Her bir sahnenin altında objeler olacaktır ve her bir objeye bir görsel atanacaktır. Bu görsel bir 3D obje, resim veya video olabilir. Ayrıca her bir objeye animasyon tanımlanabilir. Bir animasyon objenin zaman çizelgesi boyunca bulunduğu pozisyonları belirterek tanımlanır. Böylece zaman ilerledikçe obje bu pozisyonlar arasında hareket eder ve bu bir animasyon gibi gözükür.

Sunucu kısmında Web arayüzünden gelen veriler işlenip kaydedilir ve gerektiğinde mobil uygulamaya aktarılır. Web arayüzünden gelen görseller, daha yüksek yükleme ve görselleştirme hızı sağlamak için bir paketleme adımından geçer. Bu adım sayesinde görseller mobil uygulama veya Web arayüzünde hızlıca tekrar kullanılabilir. Sahneler ve görseller önbelleğe alınır. Böylece üzerlerinde bir değişim olmadığı sürece hızlıca tekrar servis edilir.

Web arayüzünde sahne oluşturulabilir, sahneye objeler koyulabilir, objelerin pozisyonu ve animasyonu ayarlanabilir. Oluşturulan sahneler hiçbir paketleme adımına ihtiyaç duymadan Web arayüzü üzerinde önizlenebilir. Böylece kullanıcılar mobil uygulamaya ve cihaza ihtiyaç duymadan sahnelerini test edebilir. Ayrıca Web arayüzünde bulunan içerik yönetim sistemi ile görseller kütüphanesindeki görseller yönetilebilir. Yeni obje görselleri eklenebilir ve varolan görseller değiştirilebilir veya silinilebilir.

Mobil uygulama son kullanıcının kullanabileceği bir AG uygulamasıdır. Normal bir AG uygulaması gibi cihaz kamerasını kullanarak artırabileceği bir hedef arar. Hedef bulunduğunda daha önce Web arayüzünden tanımlanan sahneyi, artırılmış görüntü olarak hedefin üstüne ekler. Bu hedef önceden tanımlanmış bir resim, QR kodu veya zemin düzlemi olabilir. Mobil uygulama bir sahneyi ilk görüntüleyişinde sahnenin bilgilerini sunucudan alır. Bunun için internet bağlantısına ihtiyaç duyar. Fakat sonraki görüntülemelerde sahne önbelleklendiği için sunucuya tekrar erişmeye ihtiyaç duyulmaz.

Böyle bir altyapının geliştirilmesinde bazı zorluklar vardır. Bu zorluklardan en belirgin olanı performans problemleridir. Bu performans problemleri, objelerin görselleri dinamik olarak belirlendiği için ortaya çıkar. Normal şartlarda bu görseller son kullanıcıya sunulmadan önce paketlenip, optimize edilerek servis edilir. Fakat kullanıcıdan yüklediği objeleri bu optimizasyon adımlarından geçirmesini beklemek kullanıcı deneyimini düşürecektir. Bunun için bulunan bir yöntem ‘asset bundling’ yöntemidir ve Unity’nin paketleme altyapısını sunucu tarafında kullanarak nesnelerin optimize edilmesiyle çalışır. Yapılan testlerde bu yöntemin performansa büyük katkı sağladığı gözlemlenmiştir.

Kullanıcılar kimi zaman varolan programlamaya ek olarak kendi yazdıkları betikleri eklemek isteyebilir. Bu da aşılması gereken bir diğer zorluktur. Bu problemi çözmek için altyapıya bir Javascript motoru eklenmiştir ve kullanıcılar bu motoru kullanarak kendi betiklerini uygulamaya gömebilir. Bu motor ayrıca kullanıcıların kendi arayüzlerini yazmalarına ve programlamalarına imkan tanır.

Yapılan testlerde performansa çok zarar vermeden böyle bir altyapının oluşturulabildiği görmüştür. Yapılan değişiklikler hiçbir paketleme ve dağıtma adımına gerek duymadan canlı olarak mobil uygulamada görülebiliyor. Kullanıcı testlerinde, dağıtılan mobil uygulamanın tekrar dağıtılmasına ihtiyaç duymadan içeriğinin değiştirilebilmesinin, değerli bir özellik olduğu anlaşılmıştır.

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. People's interest on the AR technologies increase day by day [2]. Users immerse more deeply and become more engaged with activities which are reinforced with AR. As a result, many traditional activities are being adapted to AR. In fact, AR has proven to be an efficient tool in education, training, industry, healthcare, tourism and entertainment applications [3].

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 [4]. The AR mentioned in this thesis' context 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 are also in the scope of this thesis. These devices provide some extra features such as built-in hand tracking and voice commands, but they follow the 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 viewers position and movement, before displaying virtual objects in the environment relative to the viewer [5]. 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 [6]. 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 [7]. 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 [8]. To transition AR into becoming mainstream, authoring tools that allow non-programmers to create their own AR applications is crucial [9]. 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 [10].

Unity game engine is the leading tool used in the majority of AR applications. In fact, 60% of AR applications are developed in Unity [11]. Although the cost of finding people capable of developing applications in Unity is high, Unity 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 may still be hard to use without significant Unity experience and technical expertise.

In this thesis, to make the development process smoother and easier for non-technical people, *ARgent*, an AR authoring tool built with Unity 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.

As an experiment, a marketing application will be created using *ARgent*. The simplicity *ARgent* provides and challenges faced with it during the creation of this

experimental application will be discussed. As *ARgent* has limitations, its advantages and disadvantages over other AR authoring solutions will be surveyed. Asset loading speed and other performance metrics will also be investigated and discussed.

In Section 2, literature works about AR authoring tools will be presented and their benefits and shortcomings will be discussed, and they will be evaluated from a user experience perspective. In Section 3, the implementation details of *ARgent* will be described along with the application architecture, content generation pipeline and the methods used to improve efficiency of the tool. In Section 4, user experiments conducted on *ARgent* will be presented and their results will be evaluated. In Section 5, the thesis will be concluded and future plans to improve the study will be presented.

2. LITERATURE REVIEW

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 [13]. Every software framework, language or application is an abstraction that simplifies the details of the underlying system. Essentially, the options for creating AR applications are all abstractions that hide the complexity of AR computations.

The tools and frameworks for creating AR applications can be investigated in 3 groups. Those are software development kits (SDK), game engines, and authoring tools. The options for creating AR applications will be presented in the order of increasing abstraction. SDKs have the lowest level abstraction but are more generalized, while authoring tools have the 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 thesis, this term will be used specifically when referring to Augmented Reality SDKs.

Some capabilities are essential for a good SDK. These are:

- Motion tracking, which is the process of determining the viewer's position relative to the world
- Environmental understanding, which is detecting and understanding the 3D structure of nearby objects and environment like ground, wall, tables, chairs etc.
- Light estimation, which aim 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[1] 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 [14]:

- Motion tracking
- Environmental understanding
- Light estimation
- Multiple face tracking
- People occlusion
- Motion capture

ARCore

ARCore[2] 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 [16]:

- Motion tracking
- Environmental understanding

[1] https://developer.apple.com/augmented-reality/
[2] https://developers.google.com/ar

- Light estimation

AR.js

AR.js[3] 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[4] and ARToolkit[5].

The main advantages of AR.js are [20]:

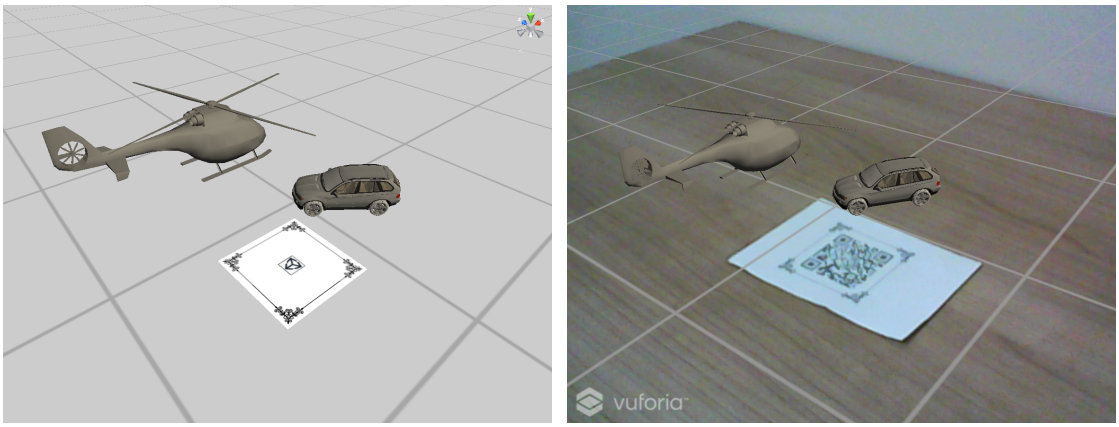
- Compatibility across browsers and devices with WebGL and WebRTC features
- Showing high performance even in old smartphones
- No need to install any app as it web-based
- Being open Source and free
- Working on a standard smartphone without additional hardware
- Being easy to use and get started

Vuforia

Vuforia[6] is a cross-platform, target-based SDK. It supports many platforms such as Windows, Android, IOS, 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 [7]. 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 [18].

[3]https://github.com/AR-js-org/AR.js
[4]https://threejs.org/
[5]http://www.hitl.washington.edu/artoolkit/
[6]https://developer.vuforia.com/



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

Figure 2.1 : A marker-based AR application

The target can be a planar 2D image or a 3D object. Vuforia supports various types of targets to track[7]:

- Model targets
- Image targets
- Object targets
- Cylinder targets
- Vumarks[8]
- 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 [17]. 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 seen in Figure 2.1.

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

[7]https://library.vuforia.com/content/vuforia-library/en/features/overview.html
[8]https://library.vuforia.com/articles/Training/VuMark.html

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 very 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 (OS) level, whereas AR.js can only access what Web API offers [21]. Web API must adopt more of the device's built-in AR features in order to become a satisfying experience.

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 away the details of SDK and provide a unified development environment that incorporates the SDKs for all supported hardware.

Unreal Engine

Unreal Engine[9] 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[10] 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

[9] https://www.unrealengine.com/
[10] https://unity.com/

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.

This kind of 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 thesis, *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 3.1. 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.

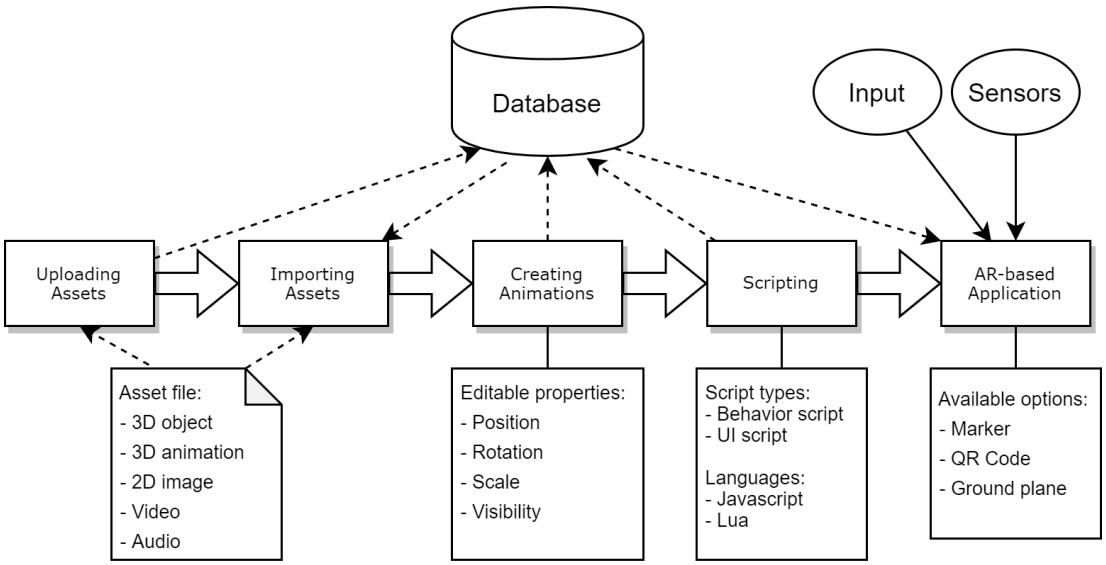


Figure 3.1 : The workflow of the proposed *ARgent* framework

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 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 3.2.

- A **scene** is a list of objects arranged to form a meaningful composition.
- An **object** is the atomic element that is shown in AR view. Every object is associated with an asset, and an animation.

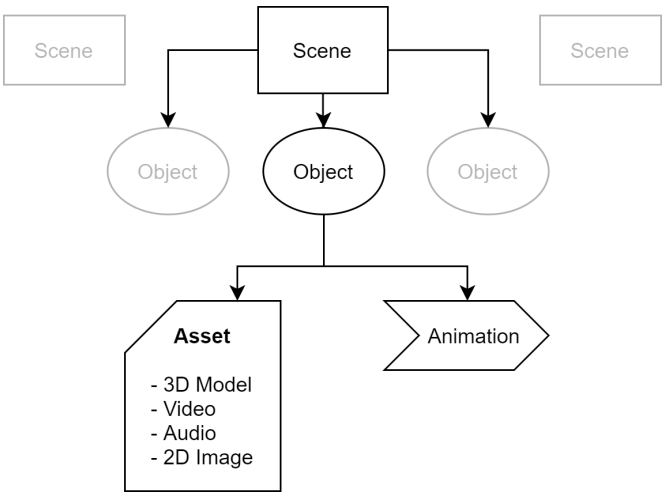


Figure 3.2 : Content structure used in *ARgent*

- 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
  - Audio

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 3.3).

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.

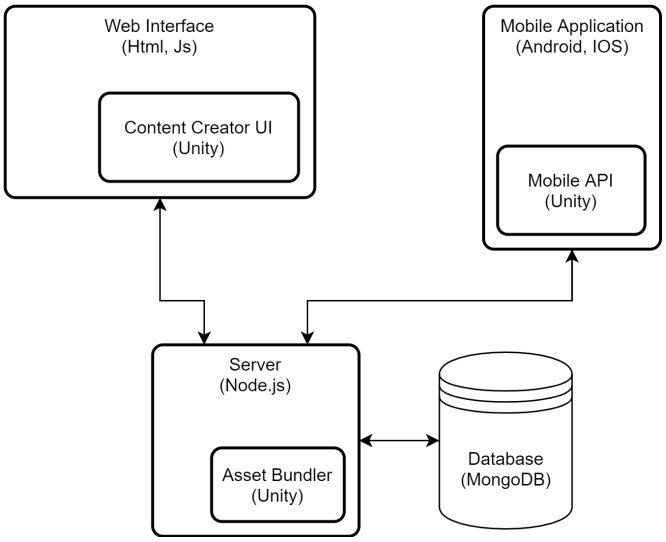


Figure 3.3 : Architecture of *ARgent* Framework

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.

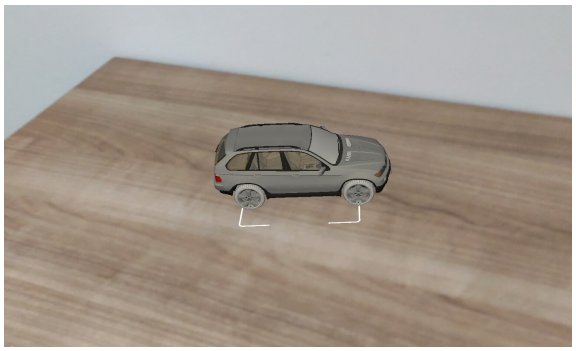
The mobile library is built in a way to be extensible and easy to integrate into other applications. The library can be included in any mobile application to add AR capabilities to the application, i.e. by wrapping it. The developers of the wrapper can call a predefined function in library with a scene identifier to spawn that scene, or let the mobile library detect scenes to show by scanning AR-code as described in Section 3.2.1.1.

In the AR view as exemplified in Figure 2.1(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 3.4, 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:

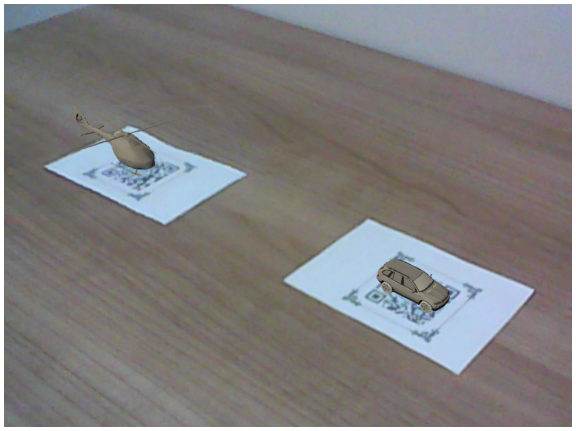
Single scene ground mode

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 3.4(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



(a) Ground mode

(b) Single marker



(c) Multiple markers with QR inside them

Figure 3.4 : Different rendering modes

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.

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 3.4(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

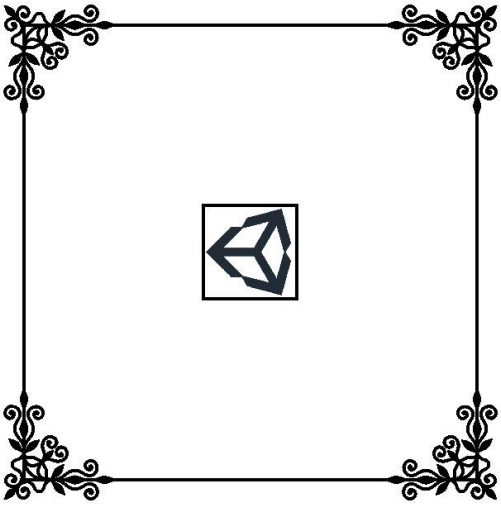
In this mode, the wrapper does not have to specify a scene identifier. The mobile library needs to detect a marker and a QR code near the marker for this method to work. When a QR code is detected near a marker, it is treated as a scene identifier and the scene is superimposed on top of the marker (Figure 3.4(c)).

One of the contributions of this thesis 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 3.5). First approach relies on the AR marker framing the QR code (Figure 3.5(b)). Second way is called a branded QR code, where QR code frames the AR marker [22] (Figure 3.5(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 [1]. Thus, the second way is not always preferable.

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



(a) Original marker

(b) QR code wrapped by marker

(c) Marker wrapped by QR code

Figure 3.5 : Difference of two ways of generating AR-codes [1]

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 3.6.

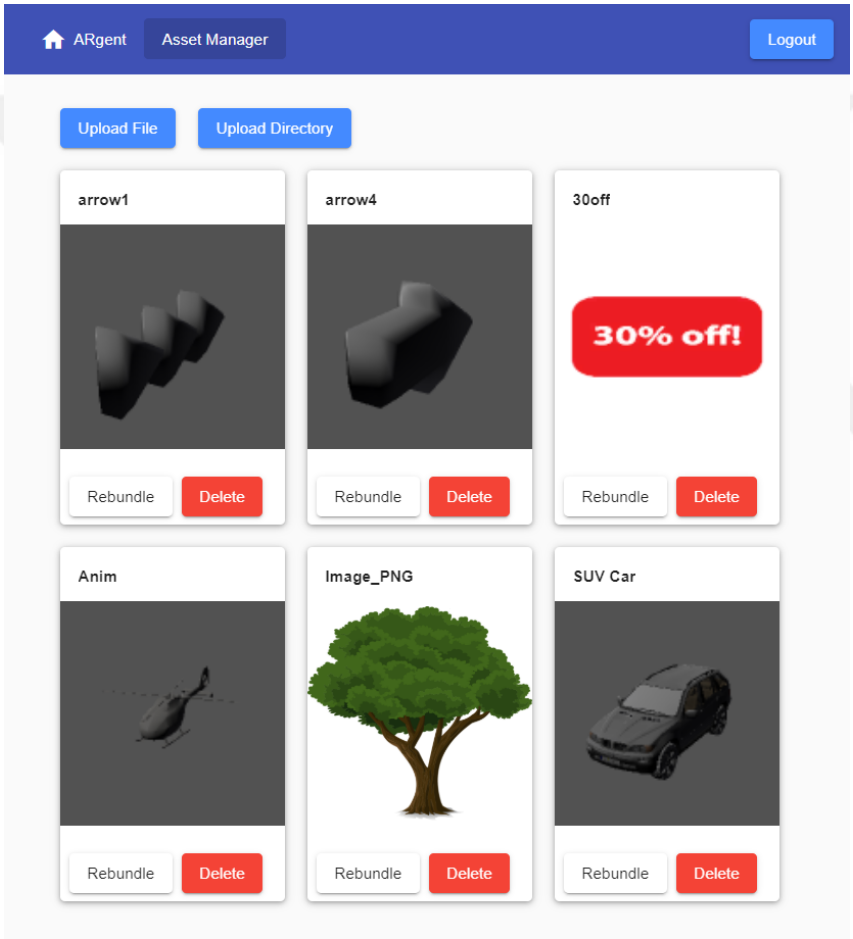


Figure 3.6 : Asset manager page

The web interface has a comprehensive UI element called the hierarchy panel (Figure 3.7). It looks and works very similar to the hierarchy and inspector panel in game engines like Unity3D. At the top of the panel ① are 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 ② or 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.

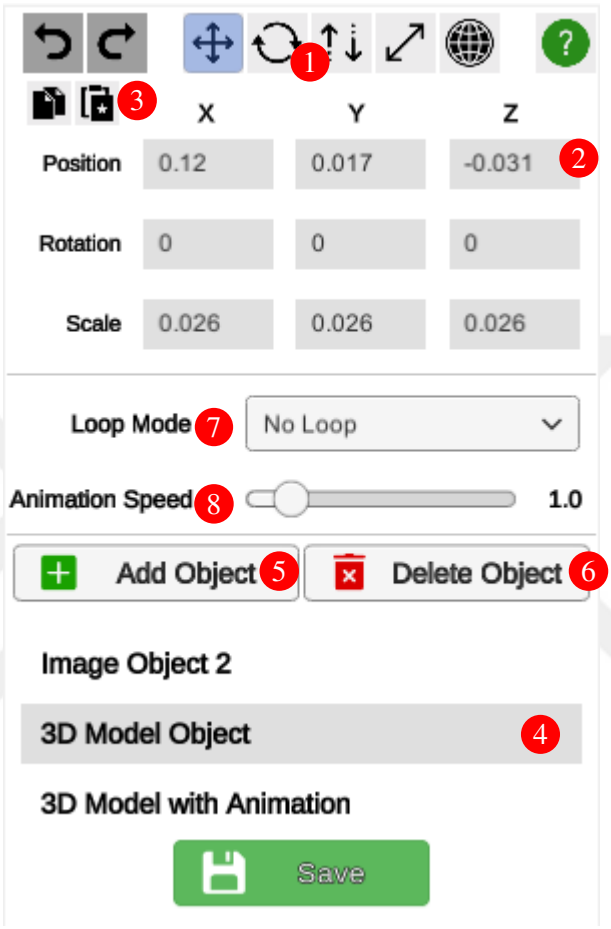


Figure 3.7 : Hierarchy panel in the web interface of *ARgent* Framework

Defining animations

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

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

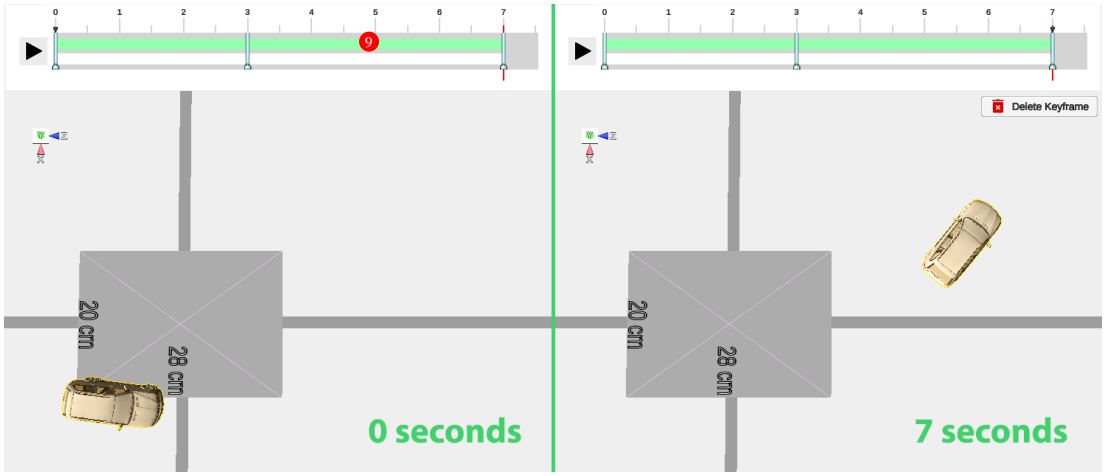


Figure 3.8 : Animation workflow in the web interface

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 3.7 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 3.7 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 3.8 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.

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 [23]. 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 regulary.

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 case study was performed. This section will describe the case study, walk the reader through the end-to-end process of creating an application with *ARgent*, and discuss the results.

4.1 Setup Phase

In the scope of this thesis, 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 4.1. 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**.

For the purpose of this application, **an image object** and **a 3D object** is imported and positioned as seen in Figure 4.1. The 3D object is an arrow pointing to the product (↓),

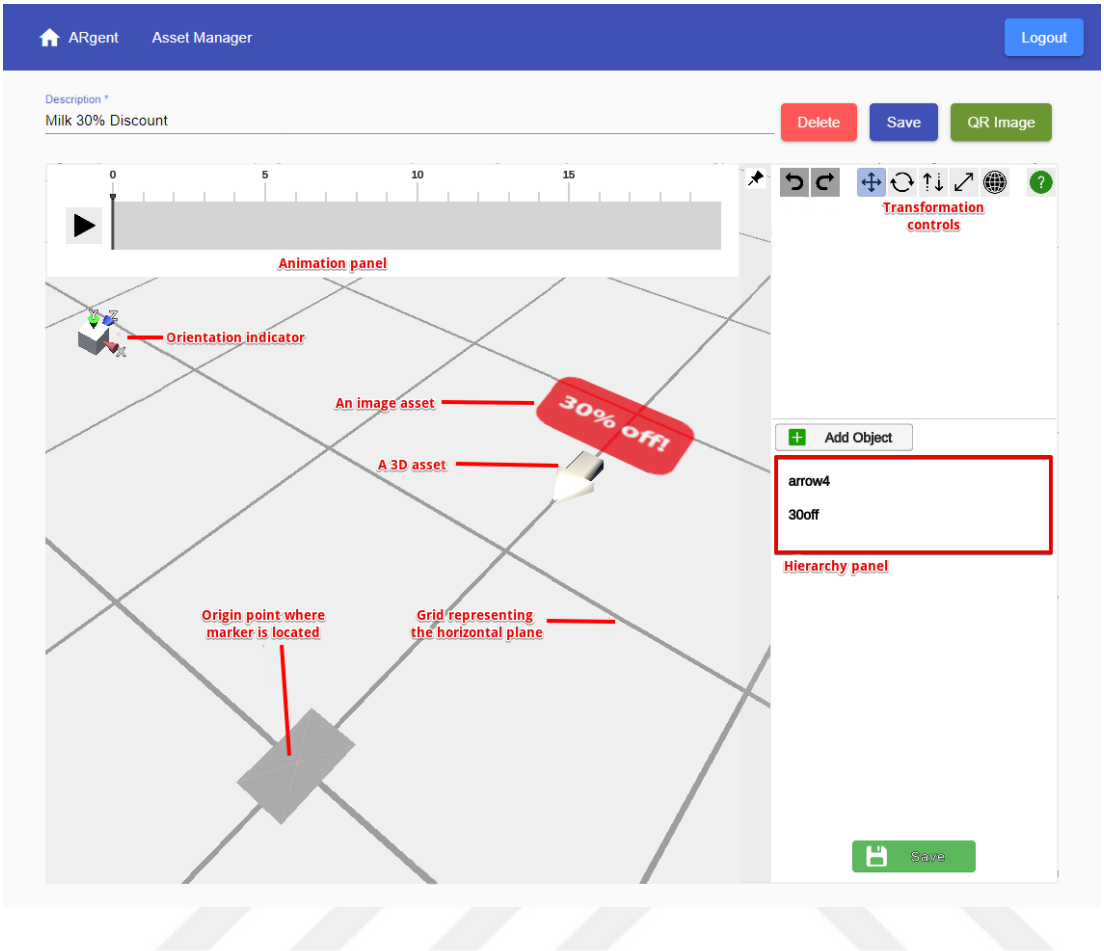


Figure 4.1 : An explanation of the web interface and scene creation process

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 4.2. The user must print-out this image and place it next to the associated product on the shelf.

4.2 Interaction Phase

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 4.3.

Figure 4.2 : AR-code for the scene created in the experimental application

Figure 4.3 : AR content shown relative to the detected marker

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 the web interface 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.

4.3 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.

Table 4.2 : Loading speeds of runtime parsing and asset bundling

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

One of the reasons 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.

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:

- The server must match all requirements of Unity
- The server must have Windows Operating System
- The server must have a GPU
- The server must have more hard-disk space
- Unity build plugins of all supported platforms, Windows, WebGL, Android, and iOS, must be installed

- A license of Unity must be set up

Opportunities 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 [24].

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. These features are not implemented yet they are considered as valuable future work.

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 experimental case 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 asset bundling method makes a lot of things possible that are normally not possible with runtime parsing method. The runtime parsing cannot handle most file formats that are common to the 3D application environment. The 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] **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.

[2] **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.

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

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

[5] **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.

[6] **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.

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

[8] **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.

[9] **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.

[10] **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.

[11] **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.

[12] **Fade, L.**, (2019), Augmented Reality In Business: How AR May Change The Way We Work, https://www.forbes.com/sites/theyec/2019/02/06/augmented-reality-in-business-how-ar-may-change-the-way-we-work, date retrieved: 31.05.2020.

[13] (2019), A Detailed Guide to Abstraction in Software with Examples, https://thevaluable.dev/abstraction-type-software-example/, date retrieved: 02.06.2020.

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

[15] **Glover, J.** (2018). *Unity 2018 Augmented Reality Projects: Build four immersive and fun AR applications using ARKit, ARCore, and Vuforia*, Packt Publishing, https://books.google.com.tr/books?id=aO1mDwAAQBAJ.

[16] **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, https://books.google.com.tr/books?id=05lUDwAAQBAJ.

[17] **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.

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

[19] **Billinghurst, M.**, **Clark, A. and Lee, G.** (2015). A survey of augmented reality.

[20] **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.

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

[22] **Hay, D.** (2012). *The Bootstrapper's Guide to the Mobile Web: Practical Plans to Get Your Business Mobile in Just a Few Days for Just a Few Bucks*, Bootstrapper's Guide, Linden Publishing, https://books.google.com.tr/books?id=6qByy15brPAC.

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

[24] (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.

[25] **Amin, D. and Govilkar, S.** (2015). Comparative study of augmented reality SDKs, *International Journal on Computational Science & Applications*, *5*(1), 11–26.

CURRICULUM VITAE

Name Surname: Gökhan Kurt

Place and Date of Birth: Turkey, 28.01.1993

Adress: Department of Computer Engineering, Istanbul Technical University, Ayazaga 34469, Istanbul-TURKEY

E-Mail: kurtgo@itu.edu.tr

M.Sc.: Game and Interaction Technologies in Istanbul Technical University, July 2020

B.Sc.: Computer Engineering in Istanbul Technical University, January 2015

PUBLICATIONS/PRESENTATIONS

- **Kurt G.**, İnce G.: Path planning in a 3D environment created using real world data. *The 23rd Signal Processing and Communications Applications Conference (SIU 2015)*, May 2016, Malatya, Turkey.

- Dube T.J., **Kurt G.**, İnce G.: An Augmented Reality Interface for Choreography Generation. Peer-reviewed, *Istanbul Journal of Innovation in Education*, January 2017, Istanbul, Turkey

- Dube T.J., **Kurt G.**, İnce G.: A Comparative Assessment of User Interfaces for Choreography Design. Peer-reviewed, *The Tenth International Conference on Advances in Computer-Human Interactions (ACHI 2017)*, March 19, 2017, Nice, France.