Proširena stvarnost na Apple IOS platformi

Permozer, Ivan

Sveučilište Jurja Dobrile u Puli
Fakultet ekonomije i turizma
„Dr. Mijo Mirković“

IVAN PERMOZER

PROŠIRENA STVARNOST NA APPLE IOS PLATFORMI

Diplomski rad

PULA 2018.

Sveučilište Jurja Dobrile u Puli
Fakultet ekonomije i turizma
„Dr. Mijo Mirković“

IVAN PERMOZER

PROŠIRENA STVARNOST NA APPLE IOS PLATFORMI

Diplomski rad

JMBAG: 0303014165, redovan
Studijski smjer: Poslovna informatika

Kolegij: Programsko inženjerstvo
Mentor: doc. dr. sc. Tihomir Orehovački

PULA, rujan 2018.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani _____, kandidat za magistra _____ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, _____, _____ godine

IZJAVA
o korištenju autorskog djela

Ja, _____ dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom _____
_____
koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu diplomskih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.
Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, _____ (datum)

Potpis

_____

SADRŽAJ

Stranica

SAŽETAK

Ubrzan tehnološki razvoj doveo je računalnu grafiku do razine pri kojoj njena primjena u svrhu proširivanja stvarnosti otvara novu perspektivu u definiranju koncepta stvarnosti. Tim slijedom pojam proširene stvarnosti predstavlja okosnicu stvarnog i virtualnog.

Milgramov kontinuum dijeli stvarnost na četiri razine: Stvarnost, proširenu stvarnost, prošireni privid i virtualnu stvarnost koje obuhvaća pod zajedničkim nazivom „mješovita stvarnost“. Koncept proširene stvarnosti počiva na kombinaciji stvarnog i virtualnog okruženja čija se interakcija u realnom vremenu poravnava na osnovu prostornih koordinata deriviranih iz procesa prikupljanja podataka. Iako virtualna i proširena stvarnost nisu sinonimi, dijele nekolicinu sličnosti. Dok virtualna stvarnost kreira umjetan svijet uz mogućnost interakcije, proširena stvarnost nudi interaktivno iskustvo nadograđujući realno okruženje korisnika.

Od začeća ideje 60-ih godina do službene upotrebe pojma početkom 90-ih godina prošlog stoljeća, proširena stvarnost utjelovljavala je ideju potencijala napredne tehnologije u svrhu unaprjeđenja i olakšanja ljudske interakcije sa okolinom, predmetima i aktivnostima.

Proširivanje stvarnosti podrazumijeva primjenu za to namijenjenog hardvera, odnosno kombinacije zaslona kao izlaznog i kamere/senzora kao ulaznog elementa. Glavna zadaća uređaja za proširivanje stvarnosti počiva na simultanom rješavanju tri osnovna problema, odnosno miješanju slike, poravnavanju virtualnih i stvarnih objekata i praćenju točaka iz realnog okruženja. Za uspješnu kombinaciju virtualnih i stvarnih scena koriste se alati kao što su *ARToolKit*, *NyARToolKit*, *FLARToolKit*, *D'Fusion*, *Qualcomm AR SDK*, *Google ARCore*, *Vuforia* i *Apple ARKit*.

Primjena proširene stvarnosti je sveprisutna, a neka od područja primjene uključuju arhitekturu, medicinu, vojne potrebe, obrazovanje, novinarstvo, marketing i zabavu. Zahvaljujući tome, danas su raspoloživa rješenja kao što su *CAMDASS* (u medicini), *BARS* (za vojne potrebe), *Construct3D* (u obrazovanju), *Layar* (u novinarstvu), *Total Immersion* razglednice (u marketingu), *FoxTrax* i srodni sustavi (u zabavne svrhe) itd.

Za kontinuirano implementiranje proširene stvarnosti u svrhu praktične primjene najzaslužniji su alati za mobilne platforme. Zahvaljujući naravi mobilnih uređaja i njihovoj svakodnevnoj upotrebi, stvorena je idealna podloga za takvu vrstu sadržaja. Slijedom toga, u sklopu *iOS* razvojnog okruženja, *Apple* developerima omogućuje razvoj aplikacija pomoću *ARKit* alata koji donosi mnoštvo pogodnosti. U sklopu *ARKit* biblioteke mogu se naći opcije kao što su mjerenje

udaljenosti dviju točaka u prostoru, prepoznavanje dvodimenzionalnih slika, te mogućnost prepoznavanja trodimenzionalnih objekata kao okidača za prikaz računalno generiranog sadržaja na *iOS 11* i novijim iteracijama *Apple iOS* operativnog sustava.

Sve češćoj implementaciji navedenih svojstava također doprinosi podrška za *Swift* programski jezik čija intuitivna narav nastoji pojednostaviti i približiti proces programiranja i razvoja aplikacija širem spektru korisnika.

Ključne riječi: Apple, AR, ARKit, Cinema 4D, Core ML, Custom Vision, Google Places, iOS, iPhone, POI, Proširena stvarnost, Mac OS, Microsoft Cognitive, Računalno učenje, Računalna vizija, SceneKit, Swift, Točke interesa, Unity, VMware, Vuforia, Xcode

ABSTRACT

Rapid advancements in technology have elevated computer graphics to the level at which its application for augmented reality purposes has spawned a new approach in regards to defining the concept of reality. Therefore the term “Augmented Reality” forms the backbone of the real and virtual worlds.

Milgram’s continuum divides reality into four different levels; Reality, Augmented Reality, Augmented Virtuality and Virtual Reality, which are commonly referred to as “Mixed Reality”. The concept of augmented reality rests on the combination of real and virtual environments whose real-time interactions are aligned based on spatial coordinates derived from the data collection process. Even though virtual and augmented reality are not synonymous, they share several similarities. While virtual reality tends to create a purely artificial world with the possibility of interaction, augmented reality offers an interactive experience by augmenting the actual environment in the eyes of the user.

Since the inception of the idea back in the 1960s all the way up to the official use of the term in the early 1990s, augmented reality embodied the potential of advanced technology for the purpose of facilitating and advancing human interactions with their surroundings, objects and activities.

Augmented reality implies the use of dedicated hardware, which consists of a screen as an output element and a camera/sensor as the input element. The main task of an augmented reality device lies in the simultaneous resolution of three main problems; overlapping images, alignment of virtual objects with the real-world environment and tracking key points from the environment. Tools such as *ARToolKit*, *NyARToolKit*, *FLARToolKit*, *D’Fusion*, *Qualcomm AR SDK*, *Google ARCore*, *Vuforia* and *Apple ARKit* are essential for a successful combination of virtual and real-world scenes.

The application of augmented reality is ubiquitous and some of it's areas of application include architecture, medicine, military purposes, education, journalism, marketing and entertainment. Thanks to this, technologies such as *CAMDASS* (in medicine), *BARS* (for military purposes), *Construct3D* (in education), *Layar* (in journalism), *Total Immersion* post cards (in marketing), *FoxTrax* and related systems (in the entertainment business) are available today.

For continuos augmented reality implementations, mobile platform tools are the most deserving when it comes to practical applications. Thanks to the nature of mobile devices and their

everyday usage, the ideal basis for that kind of content has inadvertently formed itself. Consequently, within the *iOS* development environment, *Apple's* development program enables application development using the *ARKit* tool that delivers a host of benefits. Within the *ARKit* library several features can be found, such as the ability to measure distances between two points in space, two-dimensional image recognition and the ability to detect three-dimensional objects and use them as triggers for displaying computer generated content on *iOS 11* and later iterations of *Apple's* mobile operating system.

The frequent implementation of the aforementioned features greatly benefits from the support of the *Swift* programming language, whose intuitive nature tends to simplify and demistify the process of application development for a broader spectrum of users.

Keywords: Apple, AR, ARKit, Augmented Reality, Cinema 4D, Computer Vision, Core ML, Custom Vision, Google Places, iOS, iPhone, Machine Learning, Mac OS, Microsoft Cognitive, POI, Points of interest, SceneKit, Swift, Unity, VMware, Vuforia, Xcode

1. UVOD

U dinamičnom društvu današnjice, kojem su dostupne do sada neviđene količine informacija i saznanja, pravovremena adaptacija i primjena istih potrebni su kako bi se održala efikasnost u svim granama ljudskog djelovanja. Proširena stvarnost (engl. *Augmented Reality*) jedna je od tehnologija pomoću koje je drastično izmijenjen način primjene i percepcije informacija. Iako prisutna duži niz godina, proširena stvarnost doživjela je procvat tek pojavom dostatne hardverske i softverske podrške potkraj devedesetih godina 20. stoljeća. Slijedom razvoja, spomen interakcije sa virtualnim predmetima unutar stvarnog okruženja više nije stvar fikcije, već fundamentalna stavka niza postojećih rješenja čija implementacija proširene stvarnosti nastoji unaprijediti interakciju i predstaviti pojednostavljeni način pristupa već uspostavljenim protokolima za izvršavanje privatnih aktivnosti i poslovnih projekata.

Ovaj rad osvrće se na relevantne informacije i povijesne detalje, kao i načine primjene najupečatljivijih oblika proširene stvarnosti implementiranih u sklopu komercijalnih rješenja i mobilnih aplikacija. Njihovo područje primjene obuhvaća širi spektar ljudskih djelatnosti kao što su medicina, vojne potrebe, arhitektura, edukacija, novinarstvo, marketing i zabavni sadržaj, dok mobilne aplikacije, na osnovu pristupačnijeg načina implementacije proširene stvarnosti, nastoje korisnicima pružiti spektar mogućnosti koji seže od mjerenja dimenzija predmeta u okolini do identificiranja predmeta u neposrednoj blizini korisnika.

Slijedom navedenog, pri samom početku obrađene su definicije razina stvarnosti, odnosno ključne razlike pojmova proširene i virtualne stvarnosti.

Potom, obuhvaćeni su relevantni povijesni detalji od prve primijene samog pojma do početaka suvremene upotrebe proširene stvarnosti kroz prizmu raznovrsnih ljudskih djelatnosti.

U sklopu karakteristika proširene stvarnosti obrađeni su načini miješanja slike, prostorno poravnavanje, te metode praćenja objekata čija kombinirana primjena karakterizira proizvod proširene stvarnosti.

Nadalje, pod primjenom proširene stvarnosti obuhvaćen je spektar od sedam ključnih područja ljudskog djelovanja u kojima su zabilježena neka od najupečatljivijih postignuća i načina primjene proširene stvarnosti s ciljem pojednostavljenja ljudske interakcije sa predmetima i okolinom.

Poglavlje softverske podloge osvrće se na neke od najzastupljenijih alata čija je primjena potpomogla razvoj i popularizaciju rješenja i načina primjene proširene stvarnosti u novijoj povijesti.

Na posljetku, u skladu sa ključnom tematikom, rad nastoji obuhvatiti proces i specifičnosti izrade aplikacija dizajniranih za *Apple iOS* platformu uz pomoć nekoliko za tu svrhu namijenjenih alata, te u konačnici predstaviti osvrt na funkcionalnost i pojedinosti izrađenih aplikacija analiziranih za vrijeme pokretanja i upotrebe istih na kompatibilnom uređaju.

2. RAZINE STVARNOSTI

Paul Milgram i Fumio Kishino 1994. god. postavljaju temelj Milgramovog kontinuuma. Taj virtualni kontinuum upućuje na raspon od stvarnog okruženja do potpuno virtualne stvarnosti, a klasifikacija srodnih tehnika podrazumijeva raspon od nekoliko razina (Milgram i Kishino, 1994.):

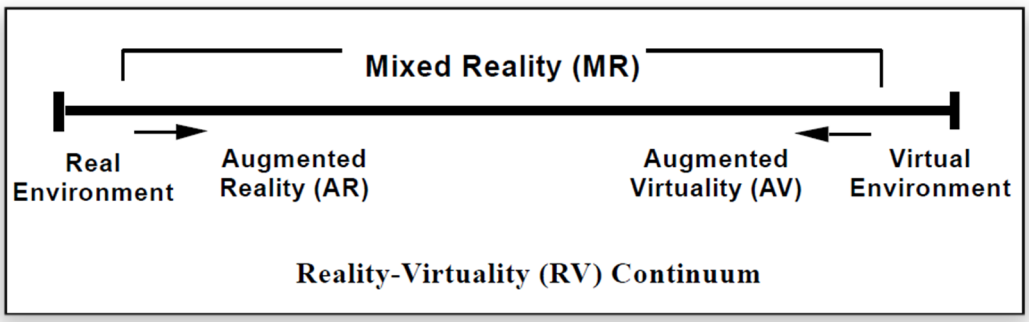
- *Real Environment* (hrv. Stvarno okruženje)
  - Korisniku je vidljivo stvarno okruženje

- *Augmented Reality – AR* (hrv. Proširena stvarnost)
  - Korisniku je vidljivo stvarno okruženje uz elemente virtualnog
  - *CGI* se mapira na specifične točke stvarnog okruženja

- *Augmented Virtuality – AV* (hrv. Prošireni privid)
  - Korisniku su vidljivi elementi stvarnog okruženja u sklopu virtualnog okruženja
  - U virtualnom okruženju se koriste snimke stvarnog okruženja

- *Mixed Reality – MR* (hrv. Mješovita stvarnost)
  - Kombinacija proširene stvarnost i proširenog privida

- *Virtual Reality – VR* (hrv. Virtualna stvarnost)
  - Korisniku je vidljivo virtualno okruženje
  - Koristi se *CGI* (*Computer Generated Imagery*) u virtualnom prostoru



Slika 1. Virtualni kontinuum

(Izvor: Milgram i Kishino, 1994., str. 3.)

2.1. Razlika proširene i virtualne stvarnosti

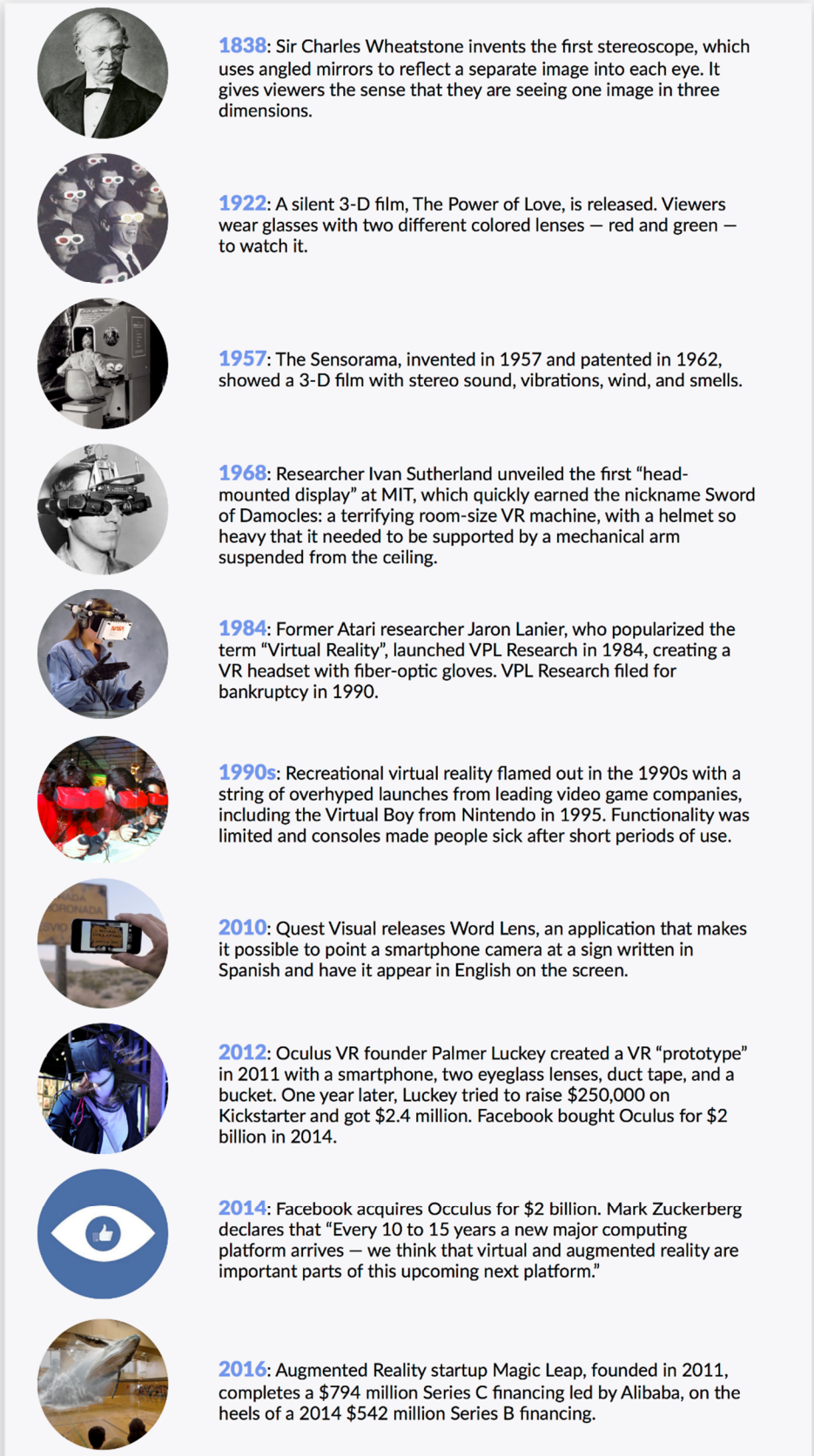
Iako virtualna i proširena stvarnost nisu sinonimi, dijele nekolicinu sličnosti. Dok virtualna stvarnost kreira umjetan svijet uz mogućnost interakcije, proširena stvarnost nudi interaktivno iskustvo nadograđujući realno okruženje korisnika.

U slučaju virtualne stvarnosti korisniku je prikazana računalno generirana dimenzija, odnosno simulirani svijet prikazan kroz spektar ljudskih osjetila kao što su vid, sluh, dodir, njuh i slično. Virtualni svijet u vidnom polju korisnika je u potpunosti umjetan, tj. svi objekti i doživljaji postoje samo unutar računalno generiranog okruženja.

Proširena stvarnost, za razliku od virtualne, iziskuje složenije tehničke specifikacije iako je realiziranje izvedbe nešto jednostavnije. Umjesto kreiranja novog okruženja u potpunosti, proširena stvarnost nastoji nadograditi postojeće okruženje korisnika pomoću primjerenih virtualnih elemenata. Stoga, u slučaju proširene stvarnosti, korisnik je u potpunosti svjestan svojeg fizičkog okruženja dok jedinu limitaciju predstavlja veličina vidnog polja na kojem se virtualiziraju elementi. Također, valja napomenuti da je za potrebe virtualne stvarnosti nezaobilazna upotreba neprovidnog *HMD* uređaja, dok je za proširenu stvarnost dostatan jedan od mobilnih uređaja novije generacije kao što je smartphone ili tablet.

Inherentna prepreka obiju tehnologija i dalje leži u dinamičnoj okolini s obzirom na nepredvidivu narav vanjskih faktora, što je posebice evidentno u razvoju proširene stvarnosti bez markera (engl. Markerless). Iako srž obiju tehnologija leži u kombiniranju stvarnih i virtualnih podražaja, proširena stvarnost je orijentirana prema primjeni u stvarnom svijetu s ciljem olakšavanja pojedinih aktivnosti korisnika, dok se narav virtualne stvarnosti pokazala pretežito popularnijom u domeni zabavnih sadržaja i gaming industrije.

Unatoč jasnoj separaciji između virtualne i proširene stvarnosti, napredak u razvoju obaju polja percipiranja stvarnosti zaslužan je za nastanak miješane stvarnosti (engl. Mixed reality) kao hibridne grane koja kombinira kvalitete spomenutih tehnologija. Miješana stvarnost je po mnogočemu slična proširenoj stvarnosti budući da se virtualni elementi manifestiraju povrh stvarnog okruženja korisnika, međutim ključna razlika leži u tome što miješana stvarnost predstavlja mogućnost međusobne interakcije virtualnog sadržaja i stvarnog okruženja u realnom vremenu. Za segment interakcije zaslužni su alati nalik senzorima pokreta i specijalnim naočalama čija se upotreba tipično spominje u kontekstu virtualne stvarnosti (Milgram, Kishino, 1994.).



Slika 2. Ilustracija vremenske crte značajnijih postignuća proširene i virtualne stvarnosti (Izvor: A2Apple, 2017.)

3. PROŠIRENA STVARNOST KROZ POVIJEST

Prvi zabilježeni spomen proširivanja stvarnosti datira iz 1896. god. Tada se, u jednom od svojih radova, psiholog George M. Stratton pozvao na *„upside-down glasses“* (hrv. Obrnute naočale) čiji je jedini efekt bilo obrtanje prikaza stvarnosti (Stratton, 1896.). Unatoč primitivnoj naravi ove naprave, može se reći da je već tada definiran smjer u kojem će se kasnije nadograđivati način percipiranja stvarnosti.

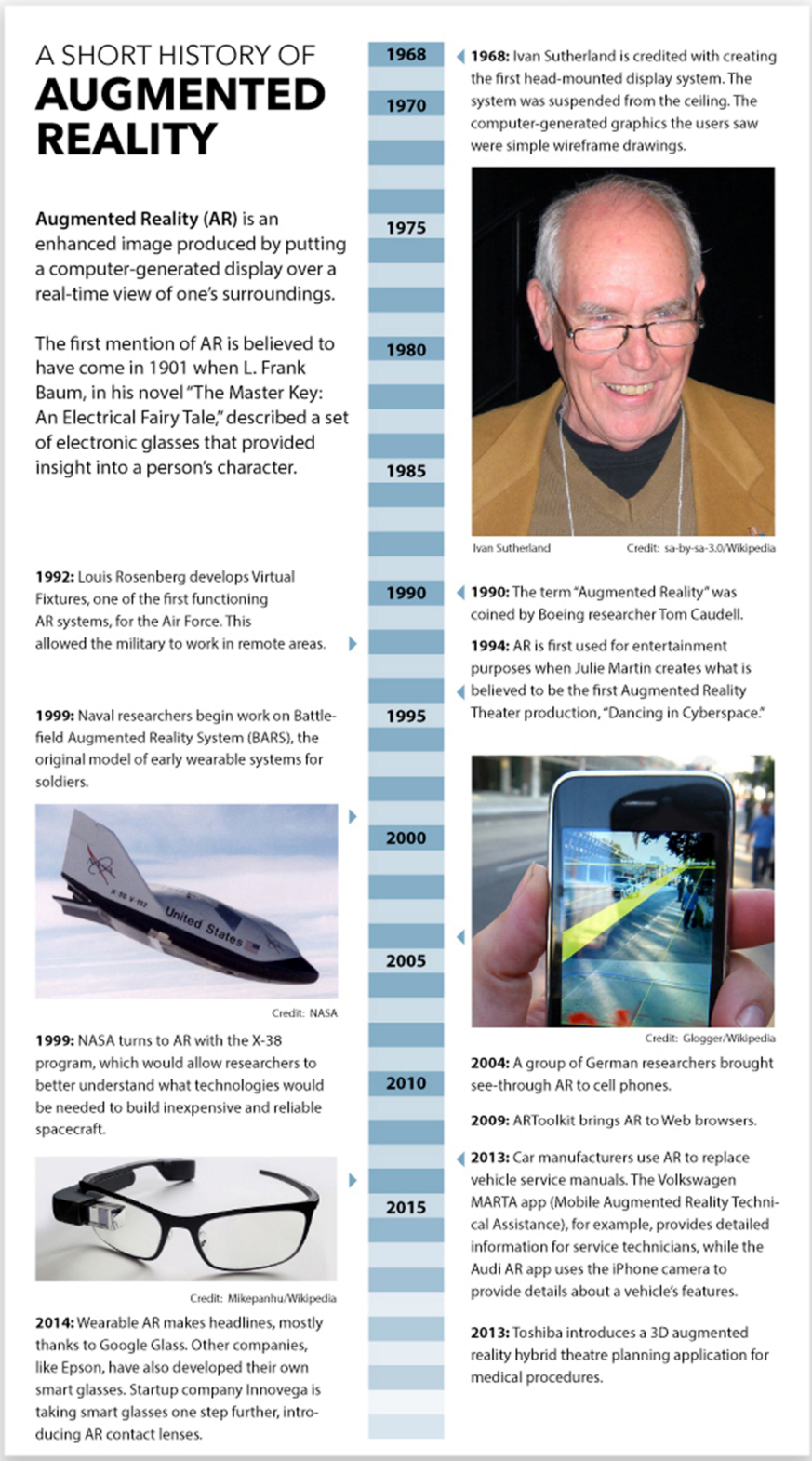
Priča proširene stvarnosti započinje 1965. god. kada je Ivan E. Sutherland u članku *„The Ultimate Display“* predstavio viziju putanje tehnološkog razvoja usmjerenu ka virtualnoj stvarnosti, *HMD* (*Head Mounted Display*) uređajima, praćenju pogleda, haptici (dodirnim podražajima), prepoznavanju govora itd. Nedugo zatim, 1968. god. predstavio je *„Sword of Damocles“* (hrv. Damaklov Mač), odnosno prototip *HMD* uređaja koji se smatra prvim sustavom virtualne i proširene stvarnosti. Sukladno limitiranim računalnim resursima tog vremena, korištena je jednostavna grafika kako bi se postigao efekt preklapanja virtualnih i stvarnih elemenata u vidnom polju korisnika, a korištenje uređaja zahtijevalo je mehaničko spuštanje sa stropa što je ujedno ukazalo na jednu od prvih mana takvog rješenja. Također, valja napomenuti kako je Sutherlandov Damaklov Mač nastao u vrijeme kada su se računala koristila isključivo u svrhe istraživanja, odnosno koncept osobnih računala je tada još bio nepoznanica (Sutherland, 1965.).

Narednih godina istraživanja u tom polju su nastavljena, no nije zabilježen značajniji pomak sve do 1980. god. kada je Steve Mann predstavio prvo prijenosno računalo s mogućnošću implementiranja grafike i teksta povrh slike stvarnog svijeta čime je omogućeno realiziranje aplikacija primjenjivih u stvarnom svijetu (Mann, 1997.).

Sam pojam *„Augmented Reality“* (hrv. Proširena stvarnost) pojavio se tek 1990. godine zahvaljujući znanstvenicima Boeing razvojnog tima. Tom Caudell i David Mizell na ideju proširene stvarnosti došli su pokušavajući, kao alternativu dijagramima i markerima, osmisliti eksperimentalni sustav koji bi radnicima prikazivao naputke o kablovima aviona, odnosno olakšao i ubrzao rad (Caudell, Mizell, 1992.).

Nadalje, tijekom 90-ih godina 20. stoljeća proširena stvarnost postaje relevantna tema. Loomis i suradnici (1993.) predstavljaju sustav za pomoć pri kretanju slijepih osoba potpomognut *GPS* uređajem. Potom, Milgram i Kishino (1994.) predstavljaju koncept virtualnog kontinuuma koji se i dalje koristi kao referenca za definiranje razina stvarnosti. Nedugo zatim, Ronald Azuma

(1997.) u svom radu *„Survey of Augmented Reality“* (hrv. Pregled proširene stvarnosti) postavlja temelje proširene stvarnosti kao grane znanosti.



Slika 3. Vremenska crta značajnijih postignuća proširene stvarnosti (Izvor: Nelson, 2014.)

4. KARAKTERISTIKE PROŠIRENE STVARNOSTI

Koncept proširene stvarnosti počiva na kombinaciji stvarnog i virtualnog okruženja čija se interakcija u realnom vremenu poravnava na osnovu prostornih koordinata deriviranih iz procesa prikupljanja podataka.

Prema Milgramu (1994.), proširena stvarnost predstavlja nadogradnju na postojeće okruženje korisnika, odnosno čini fragment kontinuuma stvarnog i virtualnog. Unatoč brojnim definicijama proširene stvarnosti, viđenja autora se isprepliću pri doticaju činjenice da ista implicira korištenje niza tehnologija koje omogućuju miješanje stvarnog i virtualnog svijeta u realnom vremenu. Time se stvarni svijet korisnika nadograđuje dodatnim informacijama koje se manifestiraju u vidu računalno generiranih prikaza koji utječu na percepciju stvarnosti i interakciju, a kompletan koncept ovisi o prostornom poravnavanju.

Slijedom navedenog, osnovne karakteristike proširene stvarnosti su:

- Kombinacija stvarnog i virtualnog
- Interakciju u stvarnom vremenu
- Prostorno poravnavanje

Ukoliko su osnovne karakteristike zadovoljene, može se konstatirati da se radi o proizvodu proširene stvarnosti. Tako se, primjerice, *compositing* (digitalno kombiniranje dviju ili više slika) filmskih scena razlikuje u odnosu na proširenu stvarnost u tome što ne sadrži komponentu interakcije u stvarnom vremenu.

Kako bi se realizirala, proširena stvarnost iziskuje kombinaciju miješanja slike, poravnavanja i prikupljanja podataka.

4.1. Miješanje slike

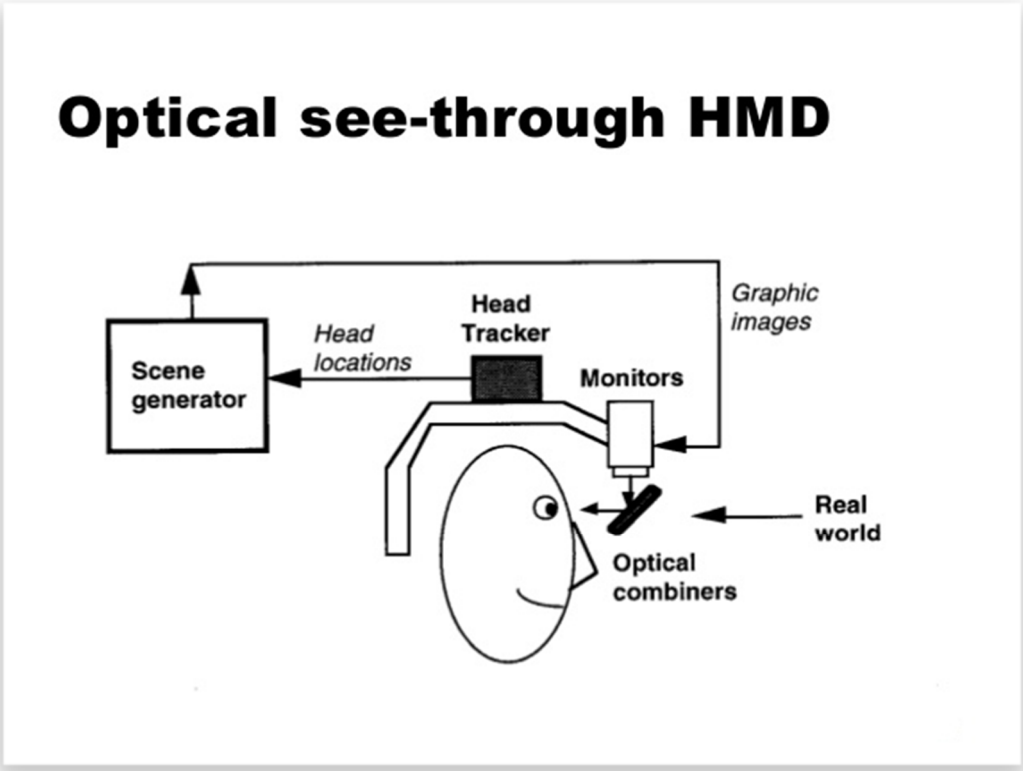
Ostvaruje se na jedan od četiri načina:

- Optičko miješanje – *Optical See Through* (*OST*)
- Video miješanje – *Video See Through* (*VST*)
- Proširena stvarnost na zaslonu – *Monitor Augmented Reality* (*MAR*)

- Projekcijska proširena stvarnost – *Projection based Augmented Reality* (*PAR*)

4.1.1. Optičko miješanje (*OST*)

Ovakav način miješanja stvarnog i virtualnog okruženja od korisnika iziskuje nošenje uređaja za prikaz pred očima (*HMD – Head Mounted Display*). Uređaj funkcionira na principu poluprozirnog ogledala pomoću kojeg je simultano vidljivo stvarno okruženje uz projekciju virtualnih elemenata koje generira uređaj. Na taj način se korisniku stvara iluzija preplitanja stvarnosti i virtualnih scena koje proizlaze iz grafički potpomognutog hardvera i projiciraju se na ogledalu u vidu dva zaslona od kojih je svaki prilagođen odgovarajućem oku. Na temelju prikupljanja podataka o poziciji glave i tijela samog korisnika, uređaj ili računalo na koje je spojen generiraju odgovarajuću perspektivu (Krevelen, Poelman, 2010.).

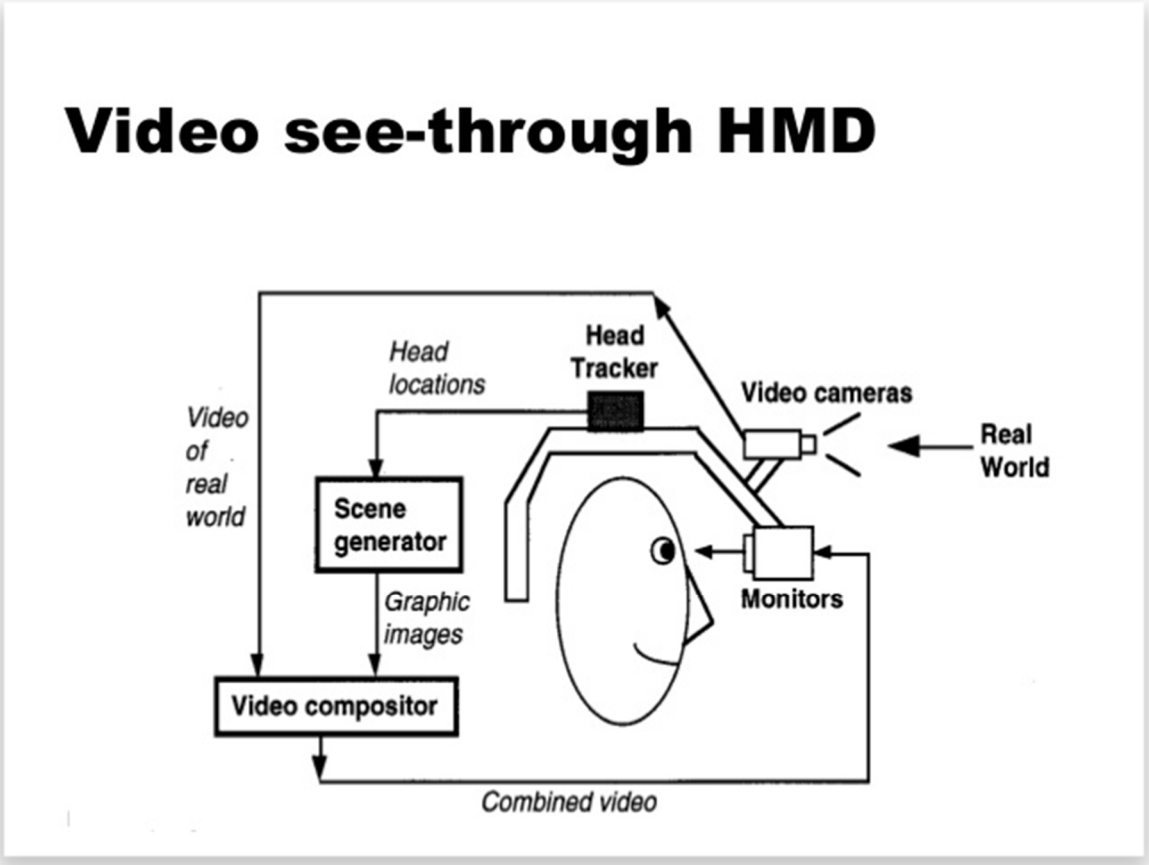


Slika 4. Koncept optičkog miješanja

(Izvor: Ganatra, 2015., str. 15.)

4.1.2. Video miješanje (*VST*)

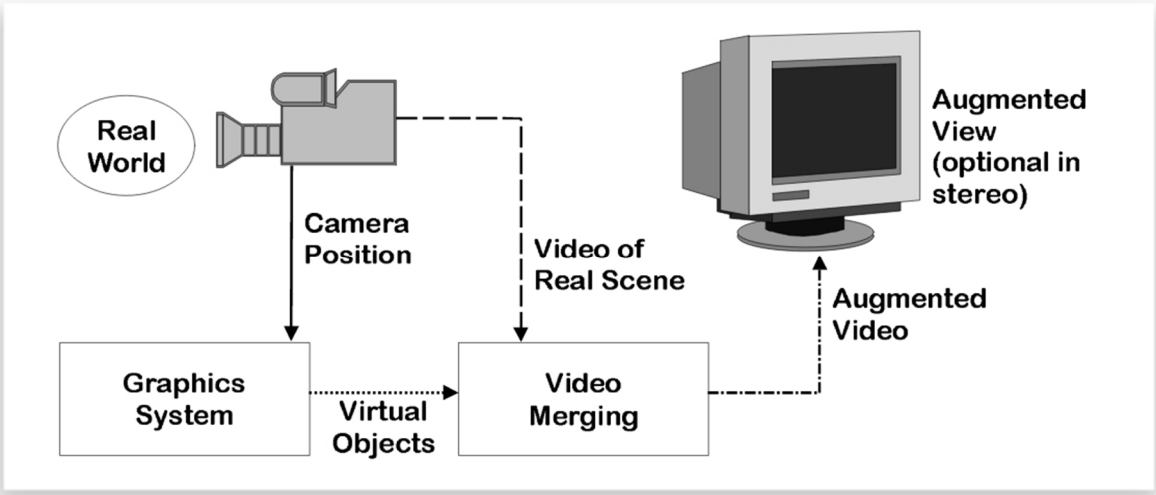
Ovaj način, nalik optičkom, također implicira korištenje uređaja koji se nalazi na glavi korisnika. Međutim, za razliku od providnog ogledala, u ovom slučaju korisniku se prikazuje okruženje percipirano putem kamera montiranih na uređaju, dok se virtualni segment generira identično optičkom miješanju. S obzirom na digitalnu narav projiciranja slike, obrada signala koji se prosljeđuje korisniku je nešto slobodnija uz dodanu mogućnost implementiranja efekata. Iz sigurnosnog aspekta, jedini propust ovakvog sustava jest činjenica da korisnik gubi vid ukoliko dođe do kvara samog uređaja budući da je jedina vizualna reprezentacija okoline dostupna putem neprovidnog zaslona (Krevelen, Poelman, 2010.).



Slika 5. Koncept video miješanja
(Izvor: Ganatra, 2015., str. 16.)

4.1.3. Proširena stvarnost na zaslonu (*MAR*)

Princip funkcioniranja ovog načina nalikuje video miješanju uz iznimku percepcije pokreta koju u ovom slučaju odrađuje kretanje kamere, a ne kretanje glave korisnika. Zahvaljujući izuzimanju *HMD* kod ovakvog načina prednost je manjak stereoskopske projekcije, odnosno na taj način se miješanje virtualnog i stvarnog okruženja prikazuje samo na jednom zaslonu (Vallino, 1998.). Jednostavnost i praktična narav ovakvog načina prikaza zaslužni su za širenje i razvoj alata za proširenu stvarnost.



Slika 6. Koncept miješanja na zaslonu

(Izvor: Vallino, 1998., str. 22.)

4.1.4. Projekcijska proširena stvarnost (*PAR*)

Kod ovog načina proširena stvarnost ne iziskuje potrebu za uređajem montiranim na glavu korisnika, već se vizualizira u realnom okruženju putem projektora. Za interakciju i praćenje pokreta zaslužne su tehnike analize, dok se projektiranje na neravne površine vrši pomoću deformiranja slike kako bi se kompenzirale anomalije površine na koju se projicira (Krevelen i Poelman, 2010.).

Slika 7. Primjer projekcijske proširene stvarnosti na *VW* automobilu
(Izvor: Patrascu, 2010.)

4.2. Prostorno poravnavanje

Poravnavanje (engl. *Registration*) u trodimenzionalnom prostoru predstavlja najveću prepreku pri realiziranju proširene stvarnosti. Problem leži u činjenici da se stvarni i virtualni elementi moraju precizno poravnati što ovisi od parametara kao što su orijentacija korisnika i položaj elemenata u sceni. Budući da je geometrijska korelacija od velike važnosti, za implementaciju virtualnih scena koristi se koordinatni sustav čije su osi usklađene sa stvarnim okruženjem (Vallino, 1998.). Za potrebe naprednijih efekata kao što su prekrivanja predmeta koristi se niz tehnika kolektivno poznatih pod nazivom *tracking* (hrv. praćenje).

4.3. Metode praćenja objekata

Praćenje objekata u svrhe proširene stvarnosti uključuje nekoliko tehnika:

- Računalni vid i obrada snimaka

- Mehaničko praćenje
- Magnetsko praćenje
- Ultrazvučno praćenje
- Inercijsko praćenje

4.3.1. Praćenje računalnim vidom i obrada snimaka

Praćenje objekata računalnim vidom podrazumijeva dva načina, odnosno praćenje pomoću markera i praćenje bez markera (Vallino, 1998.).

4.3.1.1. Praćenje pomoću markera

Unatoč većem broju načina markiranja, od ruku do *LED* dioda, jedan od najzastupljenijih oblika jest jedinstveni uzorak koji se sastoji od kontrastnih boja (najčešće crne i bijele) što omogućuje kameri lakše prepoznavanje u odnosu na okolinu. Svrha markera leži u potpori sustavu za prepoznavanje koji na osnovu prikupljenih vizualnih inputa primjereno pozicionira virtualne elemente. Prepoznavanje markera se vrši putem programske podrške, nakon čega se korisniku u vidnom polju manifestiraju virtualni elementi na pozicijama markera. S obzirom na simetričnu narav markera, programska podrška može izračunati pravilan kut prikaza sukladno perspektivnoj devijaciji percipiranog markera. Tim slijedom, virtualni objekti se mogu razmještati po trodimenzionalnom prostoru samim pomicanjem markera (Krevelen i Poelman, 2010.).

Slika 8. Primjer *AR* markera u vidu *QR* koda

(Izvor: Autor)

4.3.1.2. Praćenje bez markera

Markerless AR se koristi kada se radi o okolini koja nije kontrolirana, odnosno u kojoj nisu postavljeni markeri. Funkcionira na principu prepoznavanja prirodnih značajki kao što su teksture, rubovi i oštri vrhovi okoline kako bi se odredila pozicija u prostoru. Nakon prepoznavanja ključnih točaka i orijentacije, virtualni objekti se mogu prikazivati u vidnom polju korisnika nalik klasičnom praćenju s markerima (Krevelen i Poelman, 2010.).

Proširena stvarnost bez markera predstavlja temelj novijih rješenja s obzirom da ne zahtjeva kontrolirano okruženje i uvodi potencijalni niz opcija za praktičnu interakciju, kako za razvoj, tako i za osobnu primjenu. Stoga, ukoliko se savladaju ključni problemi kao što su poteškoće u prepoznavanju značajki za praćenje i adaptiranje na izmjeničnu rasvjetu, ova metoda nedvojbeno postaje primarni smjer razvoja alata za doživljaj proširene stvarnosti.

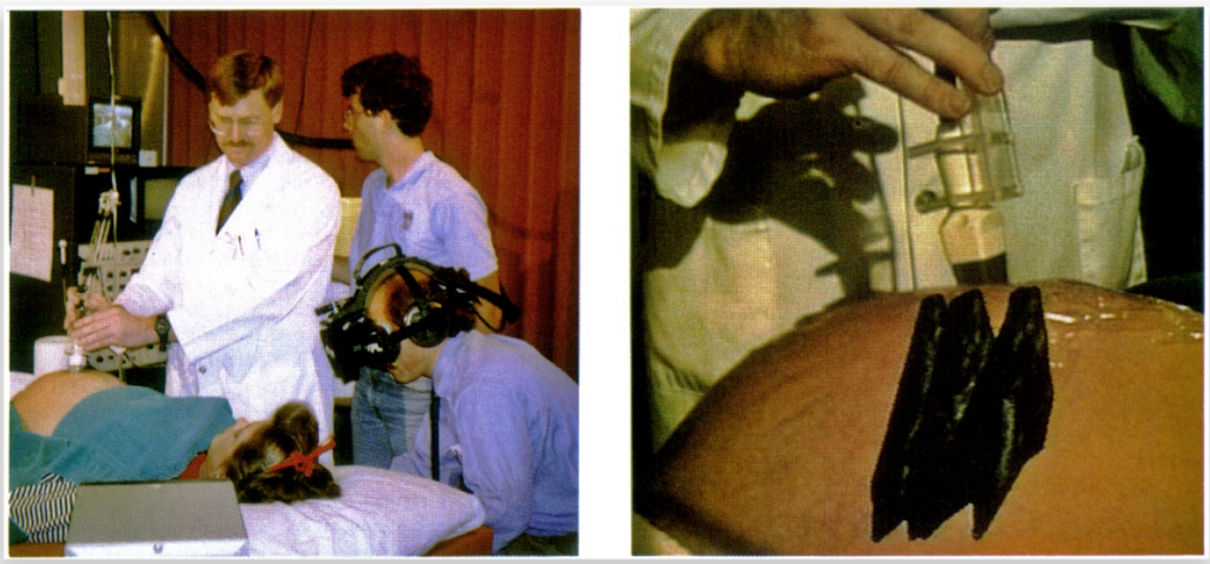
5. PRIMJENE PROŠIRENE STVARNOSTI

Unatoč prisutnosti koncepta proširene stvarnosti od 60-ih godina prošlog stoljeća, njena primjena i razvoj nastupili su tek potkraj 20. stoljeća. Razlog tomu ponajprije leži u činjenici da su u spomenutom periodu razvoj tehnologije i korisničke potražnje dosegli razinu povoljnu za potenciranje razvoja i interesa za tehnologije kao što je proširena stvarnost. Neka od prvih područja primjene proširene stvarnosti uključivala su vojne potrebe i medicinu, nakon čega je uslijedio porast interesa u područjima kao što su arhitektura, obrazovanje, novinarstvo, marketing i zabava.

Zahvaljujući sve pristupačnijem hardveru i softverskoj podlozi potrebnoj za razvoj, broj aplikacija i načina primjene proširene stvarnosti je u sve većem porastu.

5.1. Medicina

Prvi eksperiment za proširenje stvarnosti u području medicine primijenjen je 1992. godine. Efekt proširivanja stvarnosti postignut je prikazom nekoliko dvodimenzionalnih slika ultrazvuka u vidnom polju korisnika putem *HMD* uređaja. Slike su generirane pomoću *Pixel-Planes 5* sustava čija je pozicija bazirana na osnovu koordinata sustava za praćenje, a *HMD* uređaj je funkcionirao na *Video See Through* (*VST*) principu (Bajura et al., 1992.).



Slika 9. Prva dokumentirana upotreba *HMD* uređaja za proširenu stvarnost u medicinske svrhe (Izvor: Bajura et al., 1992., str. 207.)

Među novijim rješenjima nalazi se *CAMDASS* projekt čiji je razvoj potpomogla *ESA* (*European Space Agency*) potkraj 2011. godine. *CAMDASS* (*Computer Assisted Medical Diagnosis and Surgery System*) zamišljen je kao *HMD* uređaj putem kojeg bi operater mogao dobivati detaljne naputke od strane stručnog osoblja kako bi se uspješno izveo određeni zahvat medicinske naravi. Tim slijedom, smatra se da bi astronauti pomoću ovog sustava bili u mogućnosti vršiti međusobnu medicinsku dijagnostiku uz potporu medicinskog osoblja lociranog na Zemlji.

Funkcionalnost *CAMDASS* sustava zamišljena je tako da se putem ultrazvučnog uređaja i infracrvene kamere bilježe inputi i potom ispisuju u vidnom polju korisnika pomoću zaslona koji se nalazi u sklopu uređaja na glavi, a za ispravno preklapanje virtualnog sadržaja i tijela pacijenta potrebni su specifično locirani markeri. Budući da preciznost uključenih pomoćnih alata ovisi o proširenoj stvarnosti, ista zahtijeva pravilno kalibriranje za što se koristi *Single Point Active Alignment Method* (Navia et al., 2011.).

Kako bi se ustanovile prednosti *CAMDASS* sustava izvršena je evaluacija sustava u vidu praćenja dvaju grupa subjekata, od kojih je jedna izvršavala zadatak samostalno dok je druga obavlja istu radnju uz pomoć *CAMDASS* sustava. Prototip ovakvog sustava testiran je od strane hitnih službi, Crvenog križa i *Saint Pierre University Hospital* bolnice u Bruxellesu.

Slika 10. *CAMDASS* testiranje na *Saint-Pierre University Hospital*

(Izvor: ESA, 2012.)

5.2. Vojne potrebe

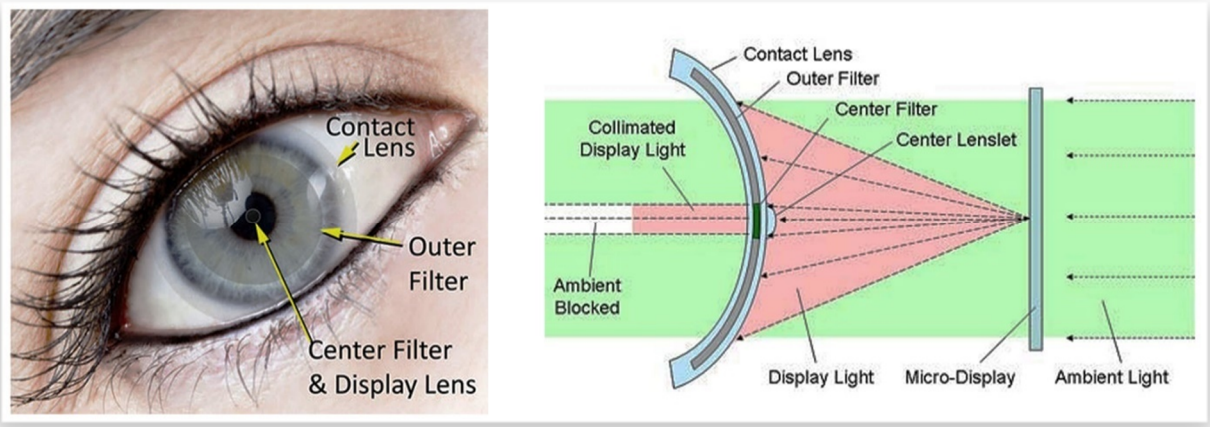
U ovoj domeni proširena stvarnost je od začeća najzastupljenija zahvaljujući velikom interesu i financijskoj potpori američkog ministarstva obrane. Primjena ovog oblika proširene stvarnosti se manifestira u dva oblika, tj. kroz nošenje uređaja u sklopu opreme vojnika na terenu i implementiranje proširene stvarnosti u sklopu vozila.

Jedan od sustava namijenjenih za korištenje vojnicima na terenu je *BARS* (*Battlefield Augmented Reality System*) predstavljen 2002. Sustav je zamišljen za upotrebu u urbanim sredinama, a neke od komponenata opreme korisnika na terenu uključuju *HMD* uređaj, prijenosno računalo i bežični uređaj za input. Pozicija i orijentacija korisnika se bilježe pomoću specifičnog uređaja za praćenje orijentacije i *GPS* jedinice, providni *HMD* zaslon je uparen sa kamerom za percepciju okoline, prikupljene informacije se pohranjuju i procesiraju putem računala pričvršćenog za korisnika, a kompletan sustav je pogonjen baterijom koja se također nalazi na korisniku. Kombinacijom tih elemenata korisniku se omogućuje prikaz okoline kroz nekoliko slojeva čija je svrha prikazati relevantne informacije kao što su obrub i oblici objekata koji bi mogli pružiti uvid u raspored prostorija za svrhe planiranja pristupa i evakuacije (Livingston et al., 2002.).

Slika 11. Elementi i prikaz *BARS* sustava
(Izvor: Livingston et al., 2002., str. 5.)

Novije iteracije sustava potpomogla je američka agencija *DARPA* (*Defense Advanced Research Projects Agency*). Nadograđena verzija *HMD* uređaja nalikuje klasičnim naočalama što je uvelike pojednostavilo način korištenja, odnosno praktičnost u kritičnim situacijama.

Nadalje, *DARPA* je za daljnji razvoj zaposlila *Innovega* tvrtku čiji je napredniji koncept predstavljen 2013. godine na *CES* konferenciji. Radi se o lećama čija bi namjena bila nalik postojećim *HMD* rješenjima koja nastoje zamijeniti u znatno manjem pakiranju (Lavaras, 2014.).



Slika 12. Koncept funkcioniranja *iOptik* sustava
(Izvor: Lavaras, 2014.)

Prvo vozilo sa implementacijom proširene stvarnosti bio je *F-35 Lightning II* borbeni zrakoplov. Projekt je rezultat zajedničkih napora dvaju izvođača vojnih radova, *RockWell Collins* i *Elbit Systems of America*, a radi se o kombinaciji najnaprednije *HMD* kacige i aviona.

Kaciga je izrađena uz pomoć *Lockheed Martin*, kompanije koja je zaslužna za stvaranje F-35 aviona, a predstavlja mnogo više od same zaštite korisnika. Kalup kacige prilagođen je korisniku na osnovu trodimenzionalnog skena glave, a tehnologije koje sadrži uključuju slušalice za prigušivanje vanjskih zvukova, sustav za noćni vid, računalni procesor u čeonom predjelu kacige i projektor koji, povrh providnog zaslona, projicira video sadržaj u realnom vremenu. Kaciga teži između 1,8 i 2,26 kg, a centar gravitacije udobno leži oko ušiju korisnika. Valja napomenuti kako kaciga funkcionira isključivo u kombinaciji sa *F-35 Lightning II* borbenim zrakoplovom, a spajanje nije bežično, već se vrši putem kablova obloženih kevlarom čija svrha je protok ulaznih i izlaznih opcija za komunikacijsku komponentu zrakoplovnog sustava.

Zahvaljujući nizu od šest kamera montiranih s vanjske strane *F-35* zrakoplova, čiji se sadržaj spaja putem *Distributed Aperture System* sustava, pilot stiče mogućnost percipiranja kompletne okoline u 360 stupnjeva uključujući i providnost samog zrakoplova. Objekti u vidnom polju pilota se omeđuju relevantnim informacijama kao što su udaljenost, brzina i visina. Također,

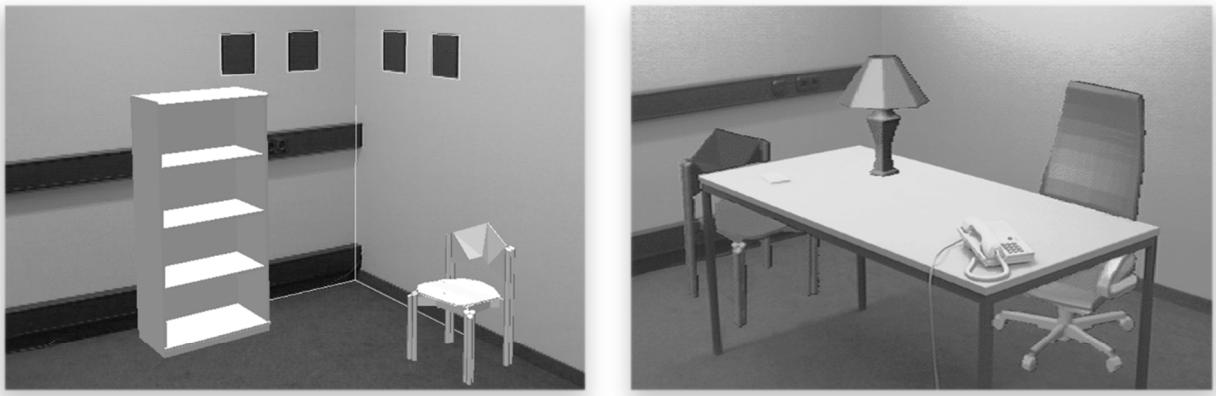
niz senzora unutar same kacige zaslužni su za precizno praćenje njene pozicije u prostoru, a oštra slika na zaslonu se projicira na osnovu precizne razdaljine između zjenica korisnika (Moynihan, 2016.).

Slika 13. Montiranje *F-35* kacige
(Izvor: Moynihan, 2016.)

5.3. Arhitektura

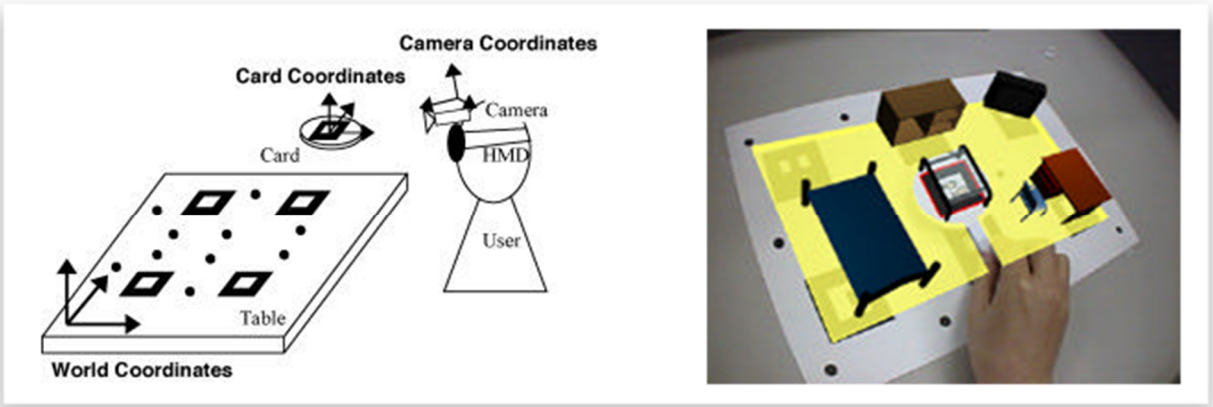
Zahvaljujući sveprisutnoj primjeni *CAD* (*Computer Aided Design*) sustava u arhitekturi implementiranje proširene stvarnosti u toj domeni bio je sljedeći logičan potez. To je rezultiralo u pojavi aplikacija zasnovanih na temelju proširenja stvarnosti iz perspektive dekoriranja eksterijera i interijera prostorija pomoću trodimenzionalnih modela namještaja.

Prvi prototip ove vrste primjene proširene stvarnosti predstavljen je 1996. godine. Breen, Whitaker, Rose i Tuceryan (1996.) u sklopu *Interactive Occlusion and Automatic Object Placement for Augmented Reality* navode kako je za primjenu takvog sustava potrebno prostoriju obilježiti odgovarajućim markerima kako bi se stekla prostorna orijentiranost. Nadalje, putem računala se vršio odabir željenih trodimenzionalnih objekata kako bi se na posljetku mogli postaviti u prostoru. Uz to postojala je i mogućnost izmijene boje interijera, odnosno zidova i podova uz primjenu odgovarajuće teksture.



Slika 14. Prvi prikaz dizajna interijera pomoću virtualnog namještaja u realnom okruženju popraćen emuliranjem umjetne rasvjete i sjena uz pomoć virtualnog objekta
(Izvor: Breen et al., 1996.)

Nešto drugačiji pristup imali su Kato, Billinghurst, Poupyrev, Imamoto i Tachibana (2000.) u sklopu *Virtual Object Manipulation on a Table-Top AR Environment*. Koncept je predstavljen 2000. godine, a zamišljen je u vidu stola čija površina je u umanjenom obliku predstavljala prostoriju za modularno izmjenjivanje nalik klasičnim arhitektonskim maketama. Virtualiziranje proizvoljno odabranih objekata se realiziralo zahvaljujući markerima na podlozi čime je korisnik mogao steći dojam kako specifičan komad namještaja, u odnosu na poziciju ostalog namještaja, izgleda u prostoriji. S obzirom na modularnu narav markera ovaj sustav se, u vrijeme začeća, smatrao jednostavnim i intuitivnim.



Slika 15. Koncept i prototip stolno baziranog sustava za proširenu stvarnost
(Izvor: Kato et al., 2000.)

5.4. Edukacija

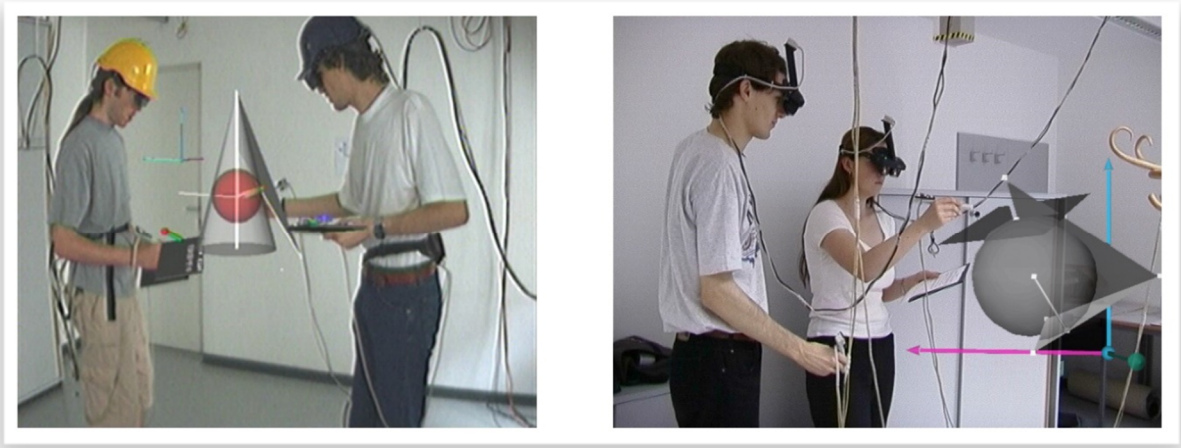
Sukladno uključivanju tehnologije u sve grane društva, edukacija nije bila iznimka. Proširena stvarnost postala je još jedna u nizu metoda za prenošenje znanja i edukaciju iz raznih aspekata ljudskih djelatnosti. Primjenom proširene stvarnosti u sklopu edukacije bi, prema znanstvenicima, predmetna materija učenicima i studentima postala shvatljivija (Yuen, 2011.).

Prema Yuen (2011.) neke od prednosti proširene stvarnosti u edukaciji obuhvaćaju:

- Poticaj učenika i studenata na samostalno proučavanje materije iz druge perspektive
- Mogućnost proučavanja sadržaja koji je inače nedostupan
- Potenciranje međusobne interakcije studenata i instruktora
- Poticaj kreativnosti
- Mogućnost prilagođavanja načina i brzine učenja sadržaja

Hannes Kaufmann je 2003. godine testirao *Construct3D* sustav na bazi matematike i geometrije koji je kombinirao proširenu stvarnost i geometrijske likove kako bi olakšao interakciju studenata i instruktora. Na taj je način postavljeno shvaćanje međusobne relacije geometrijskih likova i pripadajućih kvadranata u interaktivnom trodimenzionalnom obliku.

Construct3D, baziran na *Studierstube* sustavu, podržavao je niz jednostavnijih funkcija kao što su kreiranje točaka, linija, ploha, kvadrata, kugli i slično. Konstruiranje likova vršilo se pomoću *stylusa* (vrsta digitalne olovke), a osnovne operacije kao učitavanje, brisanje i poništavanje nalazile su se u sklopu *PIP* (*Personal Interaction Panel*) u ruci korisnika. Jedna od kasnijih iteracija uvela je glasovnu navigaciju sučelja što je ubrzalo učinkovitost samog posla (Kaufmann, 2003.).

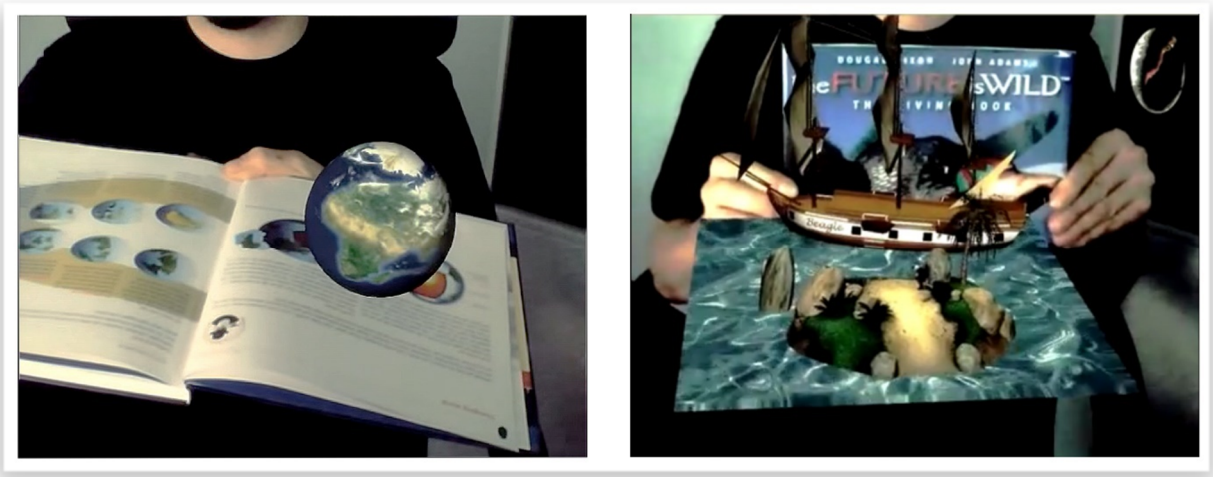


Slika 16. Međusobna suradnja studenata i interakcija sa instruktorom u *Construc3D* okruženju (Izvor: Kaufmann, Schmalstieg, 2002.)

Primjena proširene stvarnosti u edukativne svrhe se također pojavljuje i u obliku knjiga koje, uz interaktivni prikaz modela, korisniku podrobnije dočaravaju relevantne segmente sadržaja.

Početak primjene takvih projekata može se zamijetiti već od 2006. godine kada je *BBC Creative Research and Development team* inicirao razvoj podrške za proširenu stvarnost u potpori nastavnom planu Ujedinjenog Kraljevstva u sklopu interaktivnih priča za djecu. Knjige i slikovnice obogaćene su virtualnim sadržajem putem markera, a čitanje je bilo moguće putem računala (Kerawalla et al., 2006.).

Jedan od uspješnijih projekata baziranih na ovakvom principu je knjiga *The Future is Wild: The Living Book* predstavljena 2011. godine na *Frankfurt Book Fair* u Njemačkoj. Knjiga je sadržavala ukupno 42 virtualna objekta, što je rezultiralo u interaktivnoj relaciji čitatelja i knjige korištenjem Web kamere (Yuen et al., 2011.).



Slika 17. Praktični prikaz sadržaja knjige *The Future is Wild: The Living Book* (Izvor: YouTube, 2010.)

Iste godine predstavljeno je nekoliko aplikacija na sličnom principu od strane *Early-Adopter* tvrtke. Aplikacije su služile za prikaz sadržaja vezanih za nastavne predmete kao što su zemljopis i povijest. Karte i vremenske crte prožete markerima učenicima su omogućile pristup proširenom spektru informacija u obliku video sadržaja, trodimenzionalnih modela i zvučnih zapisa vezanih za specifično područje predmetne materije percipirano putem mobilnog uređaja (Early-Adopter, n.d.).

S obzirom na pozitivan aspekt proširene stvarnosti u edukaciji i sve većem broju razvojnih alata, ideja implementacije proširene stvarnosti u edukativnom sadržaju sve je izglednija, odnosno mogla bi uskoro zaživjeti na višem stupnju.

5.5. Novinarstvo

Po uzoru na implementaciju proširene stvarnosti u sklopu edukativnog sadržaja u tiskanom obliku, novinarski sadržaj srodne naravi adaptirao je identičan pristup. Tim načinom čitateljima je omogućen pristup širem spektru informacija čemu je doprinijela sveprisutna primjena mobilnih uređaja.

Koncept je zamišljen na način da, pored uobičajene kombinacije članka i slike određenog događaja, korisniku budu prikazane dodatne informacije kao što je video sadržaj koji obuhvaća ključne elemente određene reportaže koji se inače ne bi mogli obuhvatiti u tradicionalnom smislu tiskanih medija. Takav sadržaj se, po uzoru na prethodne, manifestira povrh markera raspoređenih po stranicama novina, časopisa, revija i sličnog novinarskog sadržaja.

Prvi primjer kombinacije proširene stvarnosti i novinarskog sadržaja pojavio se 2009. godine u sklopu *Esquire* magazina. Za iščitavanje virtualnog sadržaja bila je potrebna Web kamera i pripadajući računalni program, a sadržaj se manifestirao u vidu hologama zahvaljujući markerima integriranim unutar slika. Na taj način korisnicima je približen promotivni sadržaj u trodimenzionalnoj formi što je, iz perspektive potrošača, predstavilo određene prednosti (Esquire, 2009.).

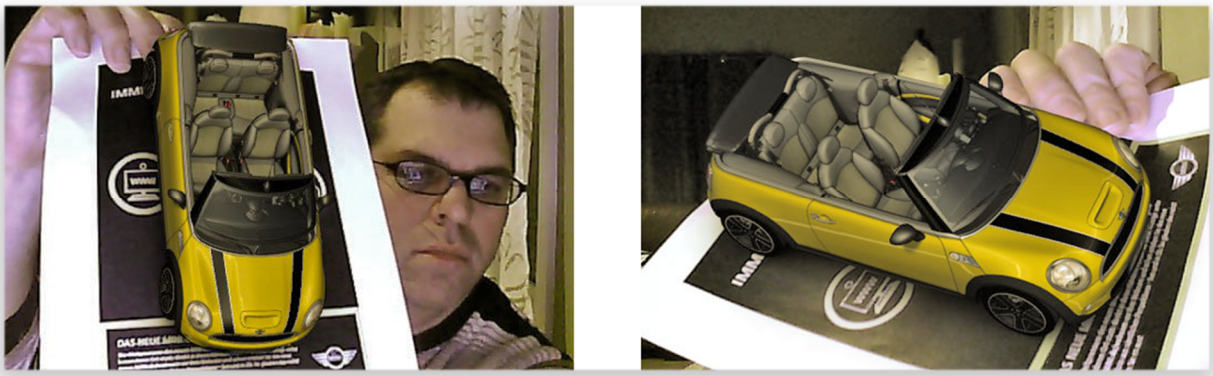
Slika 18. David Granger, urednik *Esquire* magazina, demonstrira funkcionalnost proširene stvarnosti u kombinaciji sa računalom
(Izvor: YouTube, 2012.)

Prepoznavši potencijal ovakve tehnologije, aplikacije kao što su *WorldLens* i *Layar* su taj koncept kasnije proširile izvan granica novinarskog sadržaja u vidu prijevoda stranih jezika i marketinga koristeći mobilne operativne sustave kao glavnu platformu što je nagovijestilo smjer daljnjeg razvoja i interaktivne praktične primjene proširene stvarnosti u svakodnevnom životu (Rombout, Berkel i Zakas, 2014.).

5.6. Marketing

S obzirom na uspješnu implementaciju u novinarstvu i popratnom promotivnom sadržaju, proširena stvarnost je adaptirana i unaprijeđena u marketinške svrhe. Vizualiziranje određenih predmeta prije kupnje predstavlja nedvojbenu prednost proširene stvarnosti iz perspektive potrošača, što je ujedno i jedna od primarnih ideja na kojoj je zasnovana proširena stvarnost u domeni marketinga.

Automobilska industrija je među prvima implementirala ovakav oblik marketinga, a u svojim promotivnim kampanjama ga primjenjuju kompanije kao što su *BMW, MINI*, *Ford*, *Nissan*, *Toyota*, *Mazda*, *Citroen*, *Peugeot*, *Dodge* i ostali. Tako se od 2008. godine mogu naći leci pojedinih proizvođača omeđeni odgovarajućim markerima koji, kada se percipiraju putem odgovarajućih aplikacija, prikazuju trodimenzionalne modele vozila koji su, u određenim primjerima, popraćeni glazbom i zvukovima vozila. Uz to, neki od proizvođača kao što je *Toyota* uključili su mogućnosti rastavljanja vozila na sastavne dijelove skeniranjem odgovarajućih markera, kao i mogućnost upravljanja vozilom sukladno pomicanju podloge na kojoj se nalazio ključni marker.



Slika 19. Promotivni letak tvrtke *MINI* iz 2008. god. s elementima proširene stvarnosti (Izvor: Strauss, 2008.)

Naredne godine, 2009., proširena stvarnost je prvi puta implementirana u sklopu televizijskog programa, tj. emisije *FlashForward*. Ideja je implementirana na način da korisnik pomoću isprintanog markera pristupi sadržaju koristeći Web kameru. Prilikom prepoznavanja markera na zaslonu bi se prikazivale animacije uz mogućnost proširenja sadržaja u vidu kratkih isječaka iz emisije čije ukupno trajanje je iznosilo 10 minuta. Tadašnji potpredsjednik marketinga *ABC*-a, Darren Schillace, naveo je kako se radi o hvale vrijednom postignuću budući da se na drugačiji način šira publika nikada ne bi natjerala na konzumiranje desetominutne reklame. (Kettle, 2009.)

Iste godine tvrtka *Total Immersion* predstavila je blagdanske razglednice s implementiranim virtualnim sadržajem. Sadržaj se mogao vidjeti putem odgovarajućeg softvera na računalu, a prikazan je u vidu animiranih likova popraćenih zvučnom zavjesom blagdanske tematike.

Slika 20. Izgled i prikaz sadržaja *Total Immersion* razglednice putem Web kamere (Izvor: Total Immersion, 2009.)

5.7. Zabavni sadržaj

Zahvaljujući velikom potencijalu i naravi koncepta, primjena proširene stvarnosti u zabavnoj industriji predstavila je logičan slijed. Uz mogućnost obogaćivanja već postojećeg dinamičnog sadržaja pojedinih segmenata zabavne industrije, proširena stvarnost uvela je novu dimenziju razvoja i percepcije sadržaja kao što su sportski događaji, filmovi i računalne igre.

FoxTrax sustav, predstavljen 1996. godine od strane *Fox Sports* programa, jedan je od prvih primjera implementacije proširene stvarnosti kod sportskih događaja. Sustav je korišten kako bi se putem zaslona gledateljima predočila lokacija i putanja lopte, inače teže vidljive na velikom zaslonu, kao i performanse određenih sportaša u kadru. Efekt je postignut zahvaljujući

predvidivoj podlozi sportskog terena i dresovima sportaša što je rezultiralo u prikazu relevantnih informacija u sklopu samog terena na zaslonu (SportsBusiness Journal, 2017.).

Slika 21. *Hockey puck* s integriranim odašiljačima i prikaz istog putem televizijskog prijenosa (Izvor: SportsBusiness Journal, 2017.)

Kao prvi primjer proširene stvarnosti u segmentu filmske industrije može se navesti upotreba „*3D*“ naočala. Ideja je začeta još 1922. godine, a zasniva se na stereoskopskoj percepciji, odnosno kombinaciji crvenog i zelenog providnog materijala povrh odgovarajućeg oka. Budući da je video sadržaj prikazan uz pomoć dviju filmskih vrpci, zahvaljujući čemu je obrub slike sadržavao crvene i zelene akcente, gledateljima je putem naočala omogućena iluzija trodimenzionalne slike na filmskom platnu (Symmes, 2006.). Kasnijom popularizacijom takvih filmova u razdoblju od 1950. godine nadalje korištena je kombinacija crvenog i plavog, umjesto zelenog, materijala za istu svrhu sve do 1990. godine kada je predstavljen *IMAX 3D* sustav.

Slika 22. 3D naočale iz 1924. godine (Izvor: Davis, 2014.)

Sukladno napretku tehnologije, način implementacije proširene stvarnosti u filmskoj industriji se bitno promijenio. Suvremeni primjer implementacije predstavljen je 2016. godine kao rezultat suradnje *Epic Games* proizvođača igara, produkcijske kuće *The Mill* i proizvođača motornih vozila *Chevrolet* pod nazivom *Project Raven*. Cilj projekta bio je predstaviti mogućnosti preplitanja stvarnog okruženja i vizualnih efekata u realnom vremenu čime se nastoje izbjeći učestali problemi tradicionalnog snimanja sekvenci sa zelenom pozadinom (engl. Greenscreen) i strateški pozicioniranim markerima za potrebe postprodukcijskog preklapanja sa trodimenzionalnim modelima koje se inače odvija tek nakon snimanja sekvenci. Stoga, *Project Raven* uvodi način kojim se trodimenzionalni objekti prikazuju istovremeno povrh snimke prije same postprodukcije, odnosno bez potrebe za intenzivnim procesom generiranja sekvenci sa trodimenzionalnim sadržajem povrh video materijala (engl. Compositing).

Za realizaciju projekta, odnosno preplitanje vizualnih efekata u realnom vremenu, korišteni su *Unreal Engine* tvrtke *Epic Games* i *Mill Cyclops* razvojni alat tvrtke *The Mill*. Pritom, jedino stvarno vozilo koje je snimano bilo je *Mill Blackbird*, odnosno vozilo nalik automobilu bez tradicionalne karoserije koje je bilo omeđeno markerima za praćenje na svim ključnim plohama. Podaci o poziciji i markerima bili su momentalno pohranjeni u *Unreal Engine* što je rezultiralo u trenutnoj pojavi računalno generirane fotorealne slike na zaslonu ispred redatelja, a cjelokupan kratki film *The Human Race* nastao je upotrebom navedenih tehnika. Zahvaljujući prikupljenim podacima, na osnovu jednog vozila, generirana su dva, odnosno *Chevrolet Camaro ZL1* i *Chevrolet FNR* konceptno vozilo, a povrh toga ponuđena je i mogućnost trenutnog mijenjanja tipa i boje vozila u realnom vremenu putem pripadajuće *Unreal Engine* aplikacije.

Angus Kneale, izvršni direktor *The Mill* tvrtke u New Yorku, navodi kako se radi o revolucionarnoj kombinaciji koja kroz preplitanje filmskog načina pripovijedanja i vizualnih efekata u realnom vremenu predstavlja novu eru narativnih mogućnosti, odnosno prekretnicu u polju vizualnih efekata filmske industrije i proširene stvarnosti u produkciji (Alvarez, 2017.).

Slika 23. *Mill Blackbird* (lijevo) sirova snimka, *Chevrolet Camaro ZL1* i *Chevrolet FNR* (desno) u kombinaciji sa *Unreal Engine* aplikacijom u realnom vremenu
(Izvor: Alvarez, 2017.)

Zahvaljujući kombinaciji proširene stvarnosti i razvojnog alata za računalne igre može se zaključiti kako *Project Raven* predstavlja okosnicu proširene stvarnosti, filmske produkcije i računalnih igara.

Prva računalna igra u kombinaciji sa virtualnom stvarnošću pod nazivom *ARQuake* pojavila se 2000. godine. Igra je bazirana na, godinu ranije objavljenom, *ARToolKit* razvojnom alatu, a igračima je pružala mogućnost sukoba sa virtualnim likovima putem stvarnog okruženja. Kasnije, na temelju igre, stvorena je *AR* platforma za općenitu upotrebu pod nazivom *Tinmith-Metro*. U narednom periodu pojavio se niz igara sa proširenom stvarnošću od kojih se ističe *Bystander* putem koje su online igrači nastojali izbjeći trkače iz stvarnog svijeta. (Krevelen i Poelman, 2010.)

Sony Computer Entertainment 2003. godine predstavlja *EyeToy* komponentu koja je funkcionirala kao Web kamera, odnosno dodatak *Sony PlayStation 2* konzoli. Iako su ranije već predstavljene igre koje koriste Web kameru, *EyeToy* se smatra prvim komercijalnim uspjehom takvog proizvoda u sklopu igraće konzole. Zahvaljujući činjenici da se radilo o igraćoj konzoli, *EyeToy* je proširenu stvarnost doveo u ruke šireg spektra korisnika čime se simultano proširila svijest o mogućnostima i potencijalu proširene stvarnosti u sklopu igara.

Primjena proširene stvarnosti u sklopu mobilnih igara počela se razvijati primjenom *ARToolKit* razvojnog alata za *Symbian OS* platformu 2005. godine, od kojih je *AR Tennis* jedan od prvih primjera proširene stvarnosti uz komponentu interakcije s ostalim korisnicima. Funkcionalnost igre je zamišljena na način da dva igrača sjede nasuprot jedan drugom, dok na podlozi među njima leži papir sa markerima pomoću kojih aplikacija na mobilnom uređaju generira virtualni sadržaj, odnosno teniski teren. Na taj način je igračima omogućeno dodavanje lopte

pomicanjem mobilnog uređaja u odgovarajuću poziciju, a povratne informacije, pored vizualnih, manifestirane su putem zvukova i vibracija. Programska podloga za realizaciju ove aplikacije koristila je *ARToolKit* biblioteku za *Symbian OS* u kombinaciji sa *OpenGL ES* grafičkom bibliotekom, a testirana je na *Nokia 6630* mobilnom uređaju pri brzini od 7 slika po sekundi. (Billinghurst, 2006.)

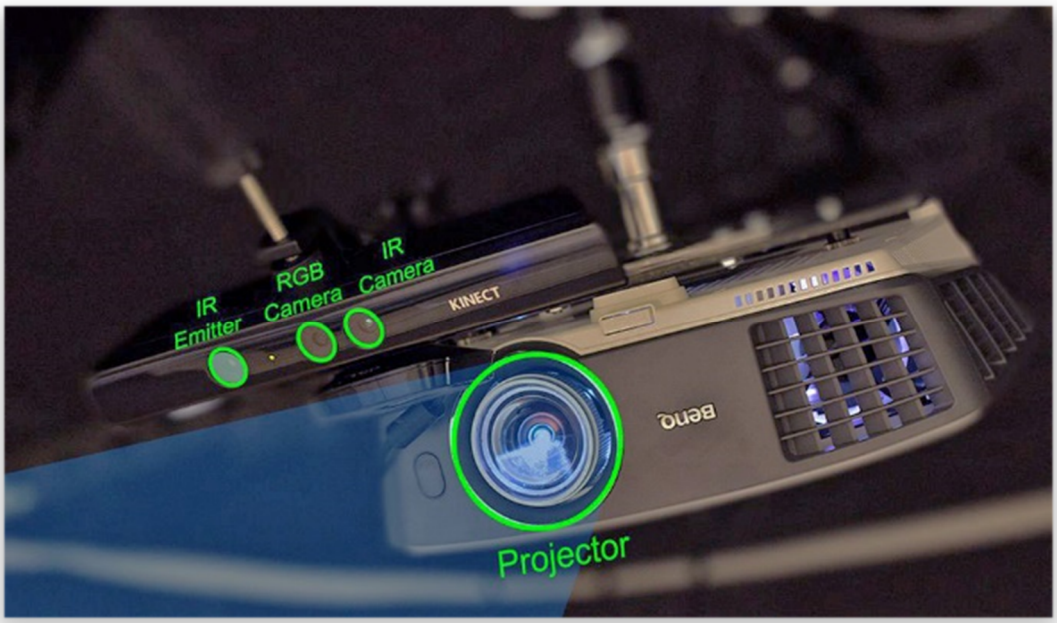
Slika 24. *Nokia 6630* i *AR Tennis* aplikacija u primjeni
(Izvor: Billinghurst, 2006.)

Od suvremenih implementacija proširene stvarnosti u igrama valja istaknuti *Microsoft Kinect* dodatak za *Xbox* konzolu. Prva iteracija predstavljena 2010. godine bila je namijenjena za Xbox 360 konzolu i sastojala se od tri komponente. Prva komponenta je bila *VGA* kamera, nalik tradicionalnoj Web kameri, pomoću koje je omogućeno prepoznavanje lica korisnika ili boja u prostoru kroz *RGB* (*Red Green Blue*) spektar. Nadalje, senzor za dubinu omogućavao je, putem infracrvenog projektora i monokromatskog senzora, percepciju dimenzija prostorije neovisno o uvjetima rasvjete. Niz od četiri mikrofona bio je zaslužan za razlučivanje glasova korisnika u odnosu na okolne zvukove što je omogućavalo igračima korištenje glasovnih naredbi sa veće udaljenosti od samog uređaja. Značaj prikupljenim podacima davala je softverska podloga ovog sustava putem koje je, kroz 48 točaka na pojedinom igraču, bilo omogućeno mapiranje za digitalnu reprezentaciju tijela korisnika uključujući i detalje lica što implicira korištenje *markerless* metode praćenja. Na osnovu točaka za praćenje sustav je mogao prepoznati korisnike neovisno o prekrivanju određenog dijela tijela ili preplitanju markera više korisnika (Crawford, n.d.).

Microsoft je 2014. godine predstavio nadograđenu verziju *Kinect* uređaja koji je namijenjen za korištenje u kombinaciji sa *Xbox One* konzolom, računalom ili tablet uređajem. Nadograđena verzija uvela je primjenu kvalitetnijih senzora i veće rezolucije uz poboljšanu mogućnost

prepoznavanja boja u mraku. Zahvaljujući tome developerima je otvoren još širi spektar mogućnosti za razvoj igara, za što je predviđen *Kinect For Windows SDK 2.0* razvojni alat. Putem navedenog alata omogućen je razvoj u *C++, C#*, *Visual Basic* i ostalim *.NET framework* orijentiranim jezicima, a njegova primjena namijenjena je za komercijalne i *Windows Store* aplikacije (Microsoft Corp., n.d.).

RoomAlive koncept, iniciran od strane *Microsoft*a, predstavio je mogućnosti proširene stvarnosti kombiniranjem *Kinect* uređaja i projektora. Cilj projekta bio je spojiti sadržaj bilo koje prostorije sa digitalnim sadržajem kako bi se korisnicima, neprimjetnim stapanjem sa njihovim neposrednim okruženjem, proširila interakcija i kompletno *Xbox gaming* iskustvo. Kasnije je na sličnom principu zasnovan *Microsoft HoloLens HMD* sustav (Microsoft Corp., n.d.).



Slika 25. *Microsoft Kinect* uređaj u kombinaciji sa *BenQ* projektorom
(Izvor: Warren, 2014.)

6. SOFTVERSKA PODLOGA PROŠIRENE STVARNOSTI

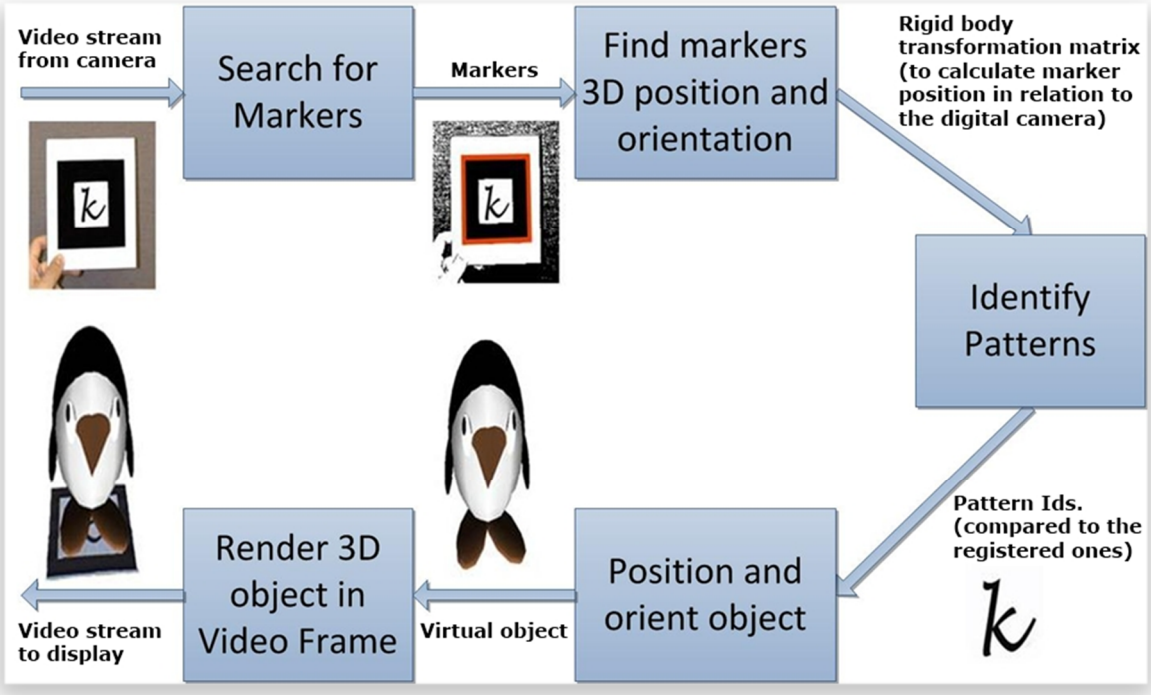
Sustavi proširene stvarnosti su, prije svega, zaduženi za praćenje, percipiranje, prikaz i interakciju. Te radnje se mogu implementirati putem softverskih okvira koji su razvijani neovisno o aplikacijama u kojima se implementiraju, a neki od najpopularnijih uključuju *ARToolKit*, *NyARToolKit*, *FLARToolKit*, *D'Fusion*, *Qualcomm AR SDK*, *Vuforia*, te najnoviji *Google ARCore* i *Apple ARKit*.

6.1. *ARToolKit*

Alat je razvio Hirokazu Kato na Nara Institutu Znanosti i Tehnologije 1999. godine, a objavljen je od strane HIT Laba na Washington Univerzitetu. *ARToolKit* je prvi puta javno predstavljen 1999. godine na *SIGGRAPH* konferenciji, nakon čega se prva *open source* inačica pojavljuje 2001. godine. Također, radi se o jednom od prvih *SDK* (*Software Development Kit*) za mobilne uređaje koji je prvi puta primijenjen 2005. god. na *Symbian OS* platformi, nakon čega su uslijedile implementacije na *iOS* uređajima 2008. i *Android* uređajima 2010. god. popraćeno službenom *ARToolWorks* profesionalnom verzijom 2011. god. (Lamb, n.d.).

ARToolKit je u srži biblioteka namijenjena za primjenu u *C* i *C++* programskim jezicima s ciljem razvoja aplikacija za proširenu stvarnost. Preklapanje stvarnog svijeta i virtualnih elemenata vrši se na osnovu specifičnih markera, a proces prikupljanja informacija uključuje:

1. Registriranje stvarnog okruženja putem kamere, odnosno video signala koji se prosljeđuje na računalo
2. Filtriranje video sadržaja s ciljem prepoznavanja visoko kontrastnog markera u vidu četverokuta (najčešće omeđen kombinacijom crne i bijele boje)
3. Izvršenje algoritma za izračun pozicije kamere u odnosu na marker (ukoliko je marker registriran)
4. Generiranje virtualnog objekta na osnovu pozicije kamere
5. Manifestiranje virtualnog objekta povrh snimke okoline čime se stvara iluzija preplitanja stvarnog i virtualnog
6. Prikaz kombinacije video materijala i generiranih virtualnih elemenata u vidnom polju korisnika što stvara dojam proširenja stvarnosti



Slika 26. Tijek rada *ARToolKit* alata

(Izvor: Barbosa et al., 2012., str. 415.)

6.2. *NyARToolKit*

Nyatla, japanski programer, 2008. godine razvio je svoju inačicu *ARToolKit* alata. Nalik *ARToolKit* alatu, *NyARToolKit* također predstavlja biblioteku funkcija za svrhe proširene stvarnosti uz ključnu razliku u tome što je namijenjen za upotrebu u kombinaciji sa nekolicinom programskih jezika kao što su *Java*, *C#*, *Actionscript*, *Processing* i *Unity*. Funkcionalnost, odnosno prepoznavanje markera i percipiranje okoline u okviru sučelja ostaju nepromijenjeni u odnosu na inicijalni *ARToolKit*, a biblioteka je dostupna za besplatno preuzimanje sa službene stranice (Virtual Worldlets Network, n.d.).

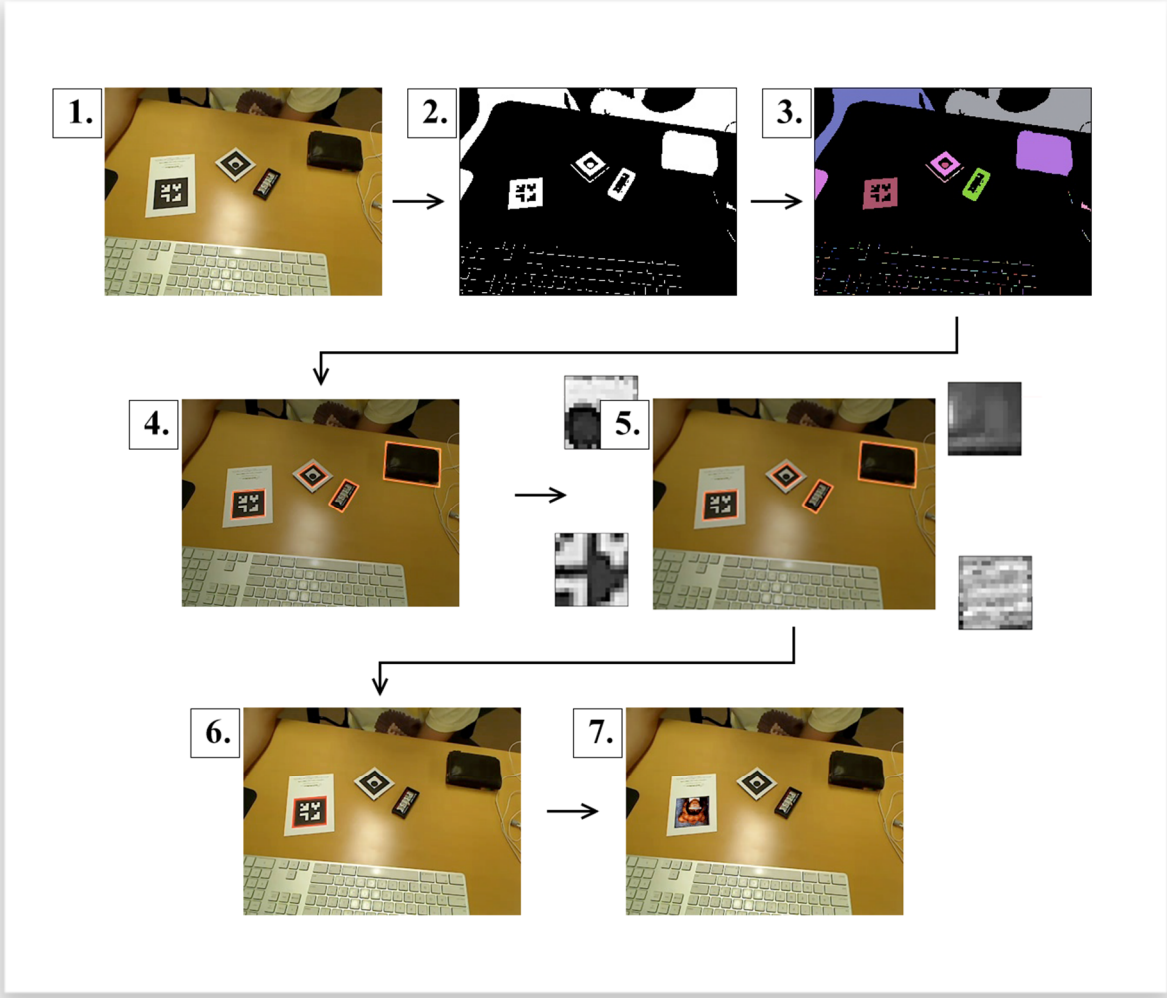
6.3. *FLARToolKit*

FLARToolKit predstavlja *Flash* inačicu, odnosno port NyARToolkit verzije biblioteke. Ova verzija je također nastala 2008. godine, proces prilagođavanja je trajao tjedan dana, a za njen nastanak zaslužan je Tomohiko Koyama. Specifičnost FLARToolKit verzije je u tome što se trodimenzionalni objekti ne generiraju u Flash okruženju, već se u navedenom jedino odvija kalkulacija pozicije markera. Nalik bibliotekama iz kojih je deriviran, *FLARToolKit*

koristi identične markere, a za generiranje objekata koriste se rješenja kao što su *Papervision3D*, *Away3D*, *Sandy*, *Alternativa3D* i sl. (Virtual Worldlets Network, n.d.).

Proces stvaranja slike proširene stvarnosti uz *FLARToolKit* se odvija na sljedeći način:

1. Registriranje slike putem Web kamere
2. Pretvaranje slike u binarni oblik
3. Označavanje
4. Pronalaženje kvadrata (markera)
5. Usklađivanje oblika i deriviranje specifičnosti markera
6. Izračunavanje matrice za preoblikovanje korištenjem algoritma
7. Prikaz trodimenzionalnih objekata



Slika 27. Praktični prikaz funkcionalnosti *FLARToolKit* biblioteke

(Izvor: Koyama, 2009.)

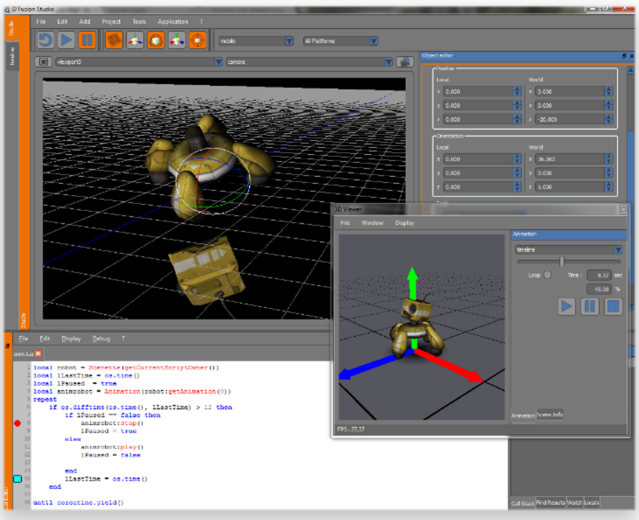
6.4. *D'Fusion*

Valentin Lefevre i Bruno Uzzan 1998. godine osnovali su *Total Immersion*, jednu od vodećih tvrtki u polju proširene stvarnosti. U početku, tvrtka je generirala većinu kapitala na osnovu fitnes simulatora pod nazivom Visio Training. Glavna značajka ovog rješenja bila je kombinacija trake za trčanje i zaslona koji je korisnicima stvarao dojam trčanja kroz ulice određenih gradova. Zahvaljujući visokoj kvaliteti i zadovoljstvu kupaca, tvrtki je popločen put za daljnji razvoj izvan okvira fitnes proizvoda.

Jedan od takvih proizvoda bio je *D'Fusion*, softversko rješenje koje se koristi za kombiniranje animacija i stvarnosti, odnosno proširenu stvarnost. Prva demonstracija ove tehnologije prikazana je kroz animaciju tigra 2004. godine, a iste godine na *DEMO Technology* konferenciji *Total Immersion* osvojio je *„Demogod“* nagradu impresionirajući skeptičnu publiku. Nedugo zatim popularnost proširene stvarnosti je doživjela nagli porast, što je iniciralo promptne inovacije i koristi proširene stvarnosti u raznim granama industrije (Total Immersion, n.d.).

Neke od *D'Fusion* karakteristika uključuju algoritme za praćenje jednog ili više markera, automatiziranu inicijalizaciju, praćenje lica, upravljanje snimkama i podršku za prepoznavanje bar kodova. Softver je dostupan za *Windows*, *Mac OS*, *Linux*, *iOS* i *Android* platforme.

Jedna od glavnih osobina *D'Fusion* sustava jest brzo prepoznavanje jednog ili više markera uz trenutni prikaz virtualnog objekta. Također, podržava i prepoznavanje djelomično prekrivenih markera što omogućuje nesmetan prikaz virtualnih objekata ukoliko je dio markera prekriven ili izvan vidokruga kamere. Valja napomenuti da je podržana izrada vlastitih markera čija jedina limitacija jest kvadratni oblik. U odnosu na *ARToolKit*, ne zahtijeva kombinaciju crno bijelih detalja u sklopu markera što proširuje funkcionalnu primjenu samog rješenja budući da se mogu koristiti i predmeti ukoliko, u vidnom polju kamere, zadovoljavaju geometrijsku formu kvadrata.



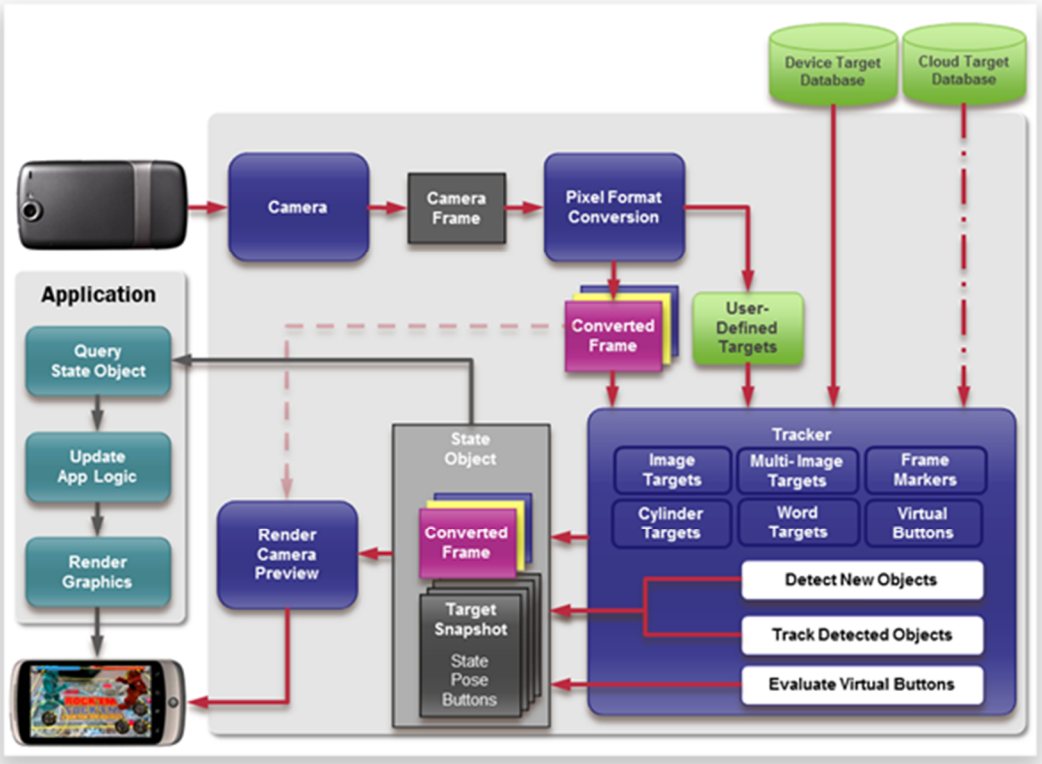
Slika 28. *D'Fusion Studio* grafičko korisničko sučelje
(Izvor: Total Immersion, n.d.)

6.5. *Qualcomm AR SDK*

Qualcomm AR SDK objavljen je 2010. godine od strane *Qualcomma* s ciljem olakšanja razvoja aplikacija proširene stvarnosti, a sam sustav revolvira oko markera dostupnih putem službenim stranica projekta. Nalik prethodnim rješenjima, *Qualcomm AR SDK* je također raspoloživ za niz platformi kao što su *Windows*, *Mac OS*, *iOS* i *Android* (Qualcomm, 2010.).

Kako bi se realizirala aplikacija bazirana na *Qualcomm AR SDK* arhitekturi potrebni su sljedeći elementi:

- **Kamera** – Zaslužna je za bilježenje i prosljeđivanje svake percipirane sekvence prema algoritmu za praćenje, dok se svaka sekvenca pohranjuje u odgovarajućoj veličini i formatu u skladu s korištenim uređajem. Za pokretanje i zaustavljanje ove komponente zaslužan je programer.
- **Konverter** – Pretvara input kamere u odgovarajući format kako bi se realiziralo praćenje i vizualiziranje objekata.
- **Tracker** (hrv. Sistem za praćenje) – Služi za lociranje i praćenje objekata pomoću algoritama zaslužnih za vid. Uz to, uključeni su i algoritmi za otkrivanje markera, dok se rezultati pohranjuju u *state object* (hrv. Objekte stanja).
- **Pozadinsko video procesiraje** – Ispisuje sekvence pohranjene u *state object*.
- **Aplikacijski kod** – U sklopu aplikacijskog koda, svaki zabilježeni isječak se zasebno procesira i potom osvježuje objekt stanja kako bi se na posljetku izvršilo ispisivanje. Ukoliko su izvršeni prethodno navedeni elementi, programer se mora pobrinuti za tri stavke:
    - Uputiti naredbu objektu stanja kako bi se ispitalo postoji li novo prepoznati objekt, odnosno je li promijenjeno stanje percipiranih markera
    - Sukladno povratnoj informaciji, ukoliko postoji promjena, potrebno je osvježiti podatke
    - Na osnovu prikupljenih parametara potrebno je vizualizirati objekte
- **Markeri** – Podrazumijevaju korištenje specifičnih datoteka generiranih putem online sustava uz mogućnost personalizacije.



Slika 29. Dijagram toka podataka *Qualcomm AR SDK* u aplikacijskom okruženju (Izvor: DeveloperIQ, 2014.)

6.6. *Vuforia*

Vuforia je platforma za razvoj proširene stvarnosti inicirana od strane *PTC Inc.* kompanije 2008. godine. Cilj projekta jest objediniti vodeće koncepte proširene stvarnosti kako bi se tvrtkama približio potencijal proširene stvarnosti u domenama kao što su proizvodnja, servisiranje, poslovanje i dizajn.

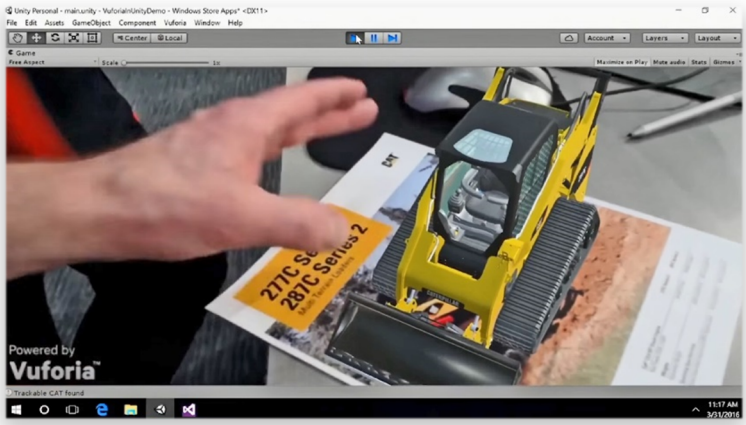
Od 2011. godine *Vuforia* tehnologija je prihvaćena od strane globalnog ekosustava koji obuhvaća više od 425,000 developera i partnera slijedom čega je razvijeno više od 50,000 aplikacija za privatne i poslovne korisnike mobilnih i *HMD* uređaja (LinkedIn, n.d.).

Danas *Vuforia* proizvodi uključuju tri ključne ponude:

- *Vuforia Engine* – za razvoj vlastitih aplikacija sa implementacijom proširene stvarnosti diljem različitih platformi
- *Vuforia Studio* – za jednostavno stvaranje i razvoj iskustava temeljenih na proširenoj stvarnosti

- *Vuforia Chalk* – za suzbijanje vremena i troškova servisa uz pomoć daljinskih mobilnih rješenja

Zahvaljujući podršci za *Unity* razvojni alat, razvoj proširene stvarnosti u sklopu aplikacija i igara za *Android* i *iOS* platforme je uvelike pojednostavljen. Unutar *Unity Asset Store* Web trgovine može se naći *Vuforia AR+VR* paket uzoraka koji sadrži nekoliko korisnih primjera za demonstraciju najvažnijih značajki *Vuforia* platforme (Unity, 2018.).



Slika 30: Demonstracija *Vuforia* aplikacije za *HoloLens HMD* putem *Unity* razvojnog alata (Izvor: Dachis, 2016.)

6.7. *Google ARCore*

Inicijalno predstavljen tijekom ožujka 2018. godine, *Google ARCore* utjelovljuje platformu za razvoj proširene stvarnosti čija funkcionalnost počiva na upotrebi niza aplikacijskih programskih sučelja (engl. *API*) na osnovu kojih uređaj stječe sposobnost osjećanja okoline, te razumijevanja i interakcije sa prikupljenim informacijama koje *Android* i *iOS* uređajima pružaju zajedničko iskustvo proširene stvarnosti (Google Inc., n.d.).

ARCore koristi tri ključne mogućnosti za integraciju virtualnog sadržaja povrh stvarnog okruženja koje podrazumijevaju:

- Praćenje pokreta – omogućuje uređaju razumijevanje i kalkulaciju vlastite pozicije u odnosu na svijet, odnosno okolinu
- Prostorno razumijevanje – pruža mogućnost detekcije veličine i pozicije svih vrsta površina koje uključuju horizontalne, vertikalne i površine pod nagibom
- Procjena svjetlosti – uređaju omogućuje procjenu svjetlosnih uvjeta u trenutnom okruženju

Po pitanju podrške, *ARCore* je dizajniran na način da obuhvaća velik broj kvalificiranih uređaja koji se baziraju na *Android 7.0*, odnosno *iOS 11* ili novijim iteracijama mobilnih operativnih sustava. Kod *iOS* verzije također valja napomenuti kako su podržani isključivo uređaji sa *A9* ili novijom arhitekturom procesora, a sama funkcionalnost limitirana je na *CloudAnchor*, dok *Android* inačica podržava *HelloAR*, *ComputerVision*, te *AugmentedImages* uzorke u sklopu *Unity* razvojnog alata.

Ukratko, funkcionalnost *Google ARCore* platforme može se svesti na dvije komponente, odnosno praćenje pozicije uređaja u pokretu, te izgradnju vlastitog shvaćanja stvarnog svijeta na temelju percipiranih parametara.

ARCore tehnologija za praćenje pokreta koristi kameru uređaja za prepoznavanje specifičnih točaka u okolini koje se potom prate i analiziraju, odnosno kombinacijom kretanja spomenutih točaka i percepcije pokreta pomoću senzora u sklopu uređaja, *ARCore* određuje poziciju i orijentaciju uređaja prilikom kretanja u prostoru.

Pored prepoznavanja točaka, *ARCore* može prepoznati i ravne površine poput stolova i podova, pri čemu također procjenjuje prosječno osvjetljenje u neposrednom okruženju. Na temelju navedenih parametara *ARCore* vrši kalkulacije, odnosno gradi vlastito shvaćanje svijeta oko sebe pomoću kojeg je korisnik u mogućnosti postavljati predmete, bilješke i relevantne informacije povrh objekata u svom okruženju, a senzori za praćenje pokreta korisniku omogućuju promatranje postavljenih virtualnih elemenata iz različitih kuteva neovisno o poziciji i kretanju uređaja (Google Inc., n.d.).

Slika 31: Primjer funkcionalnosti aplikacije bazirane na *ARCore* platformi (Izvor: Schenck, 2018.)

6.8. *Apple ARKit*

Apple je *ARKit* predstavio 2017. godine u sklopu *iOS 11* operativnog sustava. *ARKit* koristi *Visual Inertial Odometry* (procjena 3D pozicije pokretne kamere u odnosu na njenu početnu poziciju) kako bi precizno pratio svoje okruženje, što podrazumijeva kombinaciju podataka prikupljenih putem kamere i *CoreMotion* kombinacije senzora (Zia, 2016.). Time uređaj stječe mogućnost preciznog pozicioniranja unutar prostorije, bez potrebe za dodatnim kalibriranjem.

Slijedom navedenog, *iPhone* i *iPad* uređaji mogu analizirati okolinu putem ugrađene kamere i locirati ravne plohe unutar prostorije, nakon čega *ARKit* može pratiti i pozicionirati objekte na osnovu ključnih točaka. Također, korištenjem ugrađene kamere sakupljaju se i informacije o raspoloživoj rasvjeti na osnovu čega se virtualnim objektima dodjeljuje primjerena količina rasvjete i generiraju adekvatne sjene. Za generiranje takvog detaljnog i uvjerljivog virtualnog okruženja *ARKit* iziskuje *Apple A9* i *A10* procesore novije generacije s obzirom na potrebnu procesorsku snagu. Iako tek predstavljen, *Apple ARKit* je već kompatibilan s nekolicinom razvojnih alata među kojima se nalaze *Unity* i *Unreal Engine* (Apple Inc., n.d.).

Senzori koje *ARKit* koristi za funkcionalnost uključuju:

- Akcelerometar – Prikuplja informacije za sve tri osi uređaja
- Žiroskop – Služi za procjenu pozicije iz perspektive rotacionih pokreta
- Pedometar – Bilježi broj koraka u realnom vremenu, kao i prijeđenu udaljenost
- Magnetometar – Nalik kompasu, upućuje na poziciju magnetnog polja u odnosu na uređaj
- Visinomjer – Upućuje na promjene u odnosu na nadmorsku visinu

Za razliku od ostalih rješenja, *ARKit* posjeduje prednost budući da teži ka *markerless* praćenju i ne zahtijeva dodatan hardver ili kombinaciju hardvera i softvera za adekvatnu funkcionalnost. Samim time postaje pristupačan i korisnicima i developerima stoga se može očekivati nagli porast interesa glede proširene stvarnosti na *Apple* uređajima, kao i porast broja aplikacija koje će istu implementirati na inovativne načine.

Na službenoj *Apple* stranici za developere dostupna je demo verzija aplikacije koja nastoji predočiti funkcionalnost *ARKit* sučelja. Aplikacija, u trenutnoj beta verziji, sadrži pet virtualnih predmeta za prikaz: svijeću, šalicu, vazu, svjetiljku i stolicu. Izbor raspoloživih predmeta je u

korelaciji sa površinama koje su preporučene u pratećoj dokumentaciji, odnosno stolovima i podovima, koje aplikacija najefikasnije raspoznaje.

U sklopu *ARKit 1.5* verzije, objavljene paralelno sa *iOS 11.3* iteracijom operativnog sustava, uvedena je detekcija vertikalnih površina, te podrška za prepoznavanje dvodimenzionalnih slika kao što su posteri, znakovi i ilustracije. Uz navedeno, evidentna je prisutnost poboljšane sposobnosti mapiranja površina, kao i povećanje rezolucije virtualnog sadržaja za 50% (Cao, 2018.)

ARKit 2 verzija uvodi unaprijeđenu sposobnost praćenja dvodimenzionalnih slika u pokretu kako bi se omogućilo prepoznavanje objekata nalik ambalaži proizvoda i stranicama časopisa povrh kojih je moguće prikazivati računalno generirani sadržaj. Pored navedenog, implementirana je i sposobnost prepoznavanja uobičajenih predmeta kao što su skulpture, igračke i namještaj.

Sa nadograđenom verzijom *ARKit 2* biblioteke također je predstavljena i mogućnost dijeljenja doživljaja proširene stvarnosti sa većim brojem uređaja istovremeno pri čemu se ističe mogućnost nastavljanja prethodno prekinute sesije, odnosno pozicije objekata i nastale promjene ostaju zapisani u skladu sa prostornim parametrima percipiranim putem brojnih senzora u sklopu uređaja čime se u konačnici unaprjeđuje cjelokupno iskustvo interakcije sa proširenom stvarnošću (Apple Inc., n.d.).

Trenutno, pogodnosti *ARKit 2* biblioteke dostupne su isključivo developerima, a prve aplikacije koje upotrjebljuju navedena poboljšanja biti će javno dostupne sredinom rujna, odnosno u skladu sa objavom *iOS 12* iteracije operativnog sustava.

Slika 32. Test *ARKit Example* aplikacije na *iPad* uređaju (Izvor: Graham, 2017.)

7. PROŠIRENA STVARNOST NA APPLE IOS PLATFORMI

Za potrebe prikazivanja funkcionalnosti proširene stvarnosti na *Apple iOS* platformi korištena je sljedeća oprema:

- *Windows 10 Pro 1803* verzija operativnog sustava
- *VMWare Workstation 14 Pro* za potrebe virtualiziranja *Mac OS* operativnog sustava kojem su dodijeljeni sljedeći resursi:
    - Četiri procesorske jezgre na frekvenciji od 4.0 GHz
    - 8GB radne memorije
    - 80GB prostora na disku
- *Mac OS High Sierra 10.13.5* verzija operativnog sustava
- *Xcode 9.4.1* verzija sučelja za programiranje aplikacija
- *Unity 2018.2.2f1* verzija alata za razvoj igara
- *Cinema 4D R19* verzija alata za izradu trodimenzionalnih animacija
- *iPhone 7 iOS 11.4.1* uređaj

Na posljetku procesa pripreme za razvoj *iOS* aplikacija, postojećem *Apple App Store* profilu dodijeljena je developerska akreditacija čija se limitacija odnosi na maksimalan broj od tri besplatno potpisane aplikacije na korištenom uređaju, dok veći broj iziskuje plaćanje naknade.

7.1. Aplikacija za mjerenje udaljenosti

U skladu s napretkom tehnologije, mnoge praktične alate postupno zamjenjuju unaprijeđene inačice. Jedan od takvih primjera može se zamijetiti u slučaju klasičnog ravnala, odnosno vrpce za mjerenje udaljenosti čiju bi ulogu mogla preuzeti aplikacija temeljena na proširenoj stvarnosti.

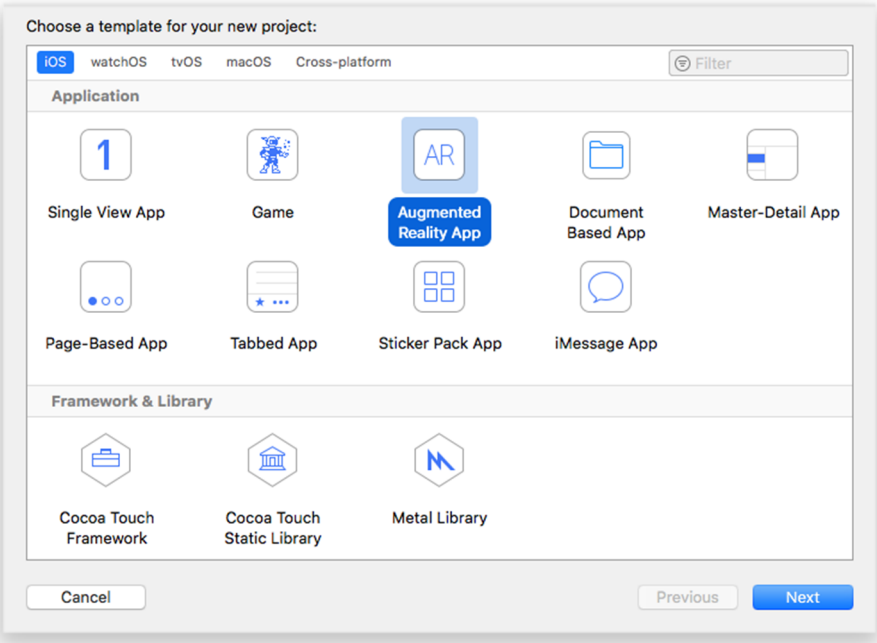
Izrađena aplikacija u nastavku nastoji ukazati na potencijal, odnosno premostiti jaz između stvarnosti i tehnologije primjenom proširene stvarnosti uz pomoć kamere kao ulaznog elementa na *iOS* uređaju u cilju ostvarenja vjerodostojnog prikaza udaljenosti dviju točaka u prostoru.

U uvodnom dijelu je obrađen kod elemenata glavnog sučelja aplikacije, dok se u nastavku obrađuju specifičnosti glede mjerenja udaljenosti dviju točaka u prostoru.

7.1.1. Kreiranje novog projekta

Prilikom kreiranja novog projekta važno je ustanoviti da li korištena verzija *Xcode* sučelja podržava *ARKit* biblioteku, odnosno poželjno je koristiti *Xcode 9* ili noviju inačicu.

Potom, za kreiranje novog projekta, odabire se *Augmented Reality APP* opcija.

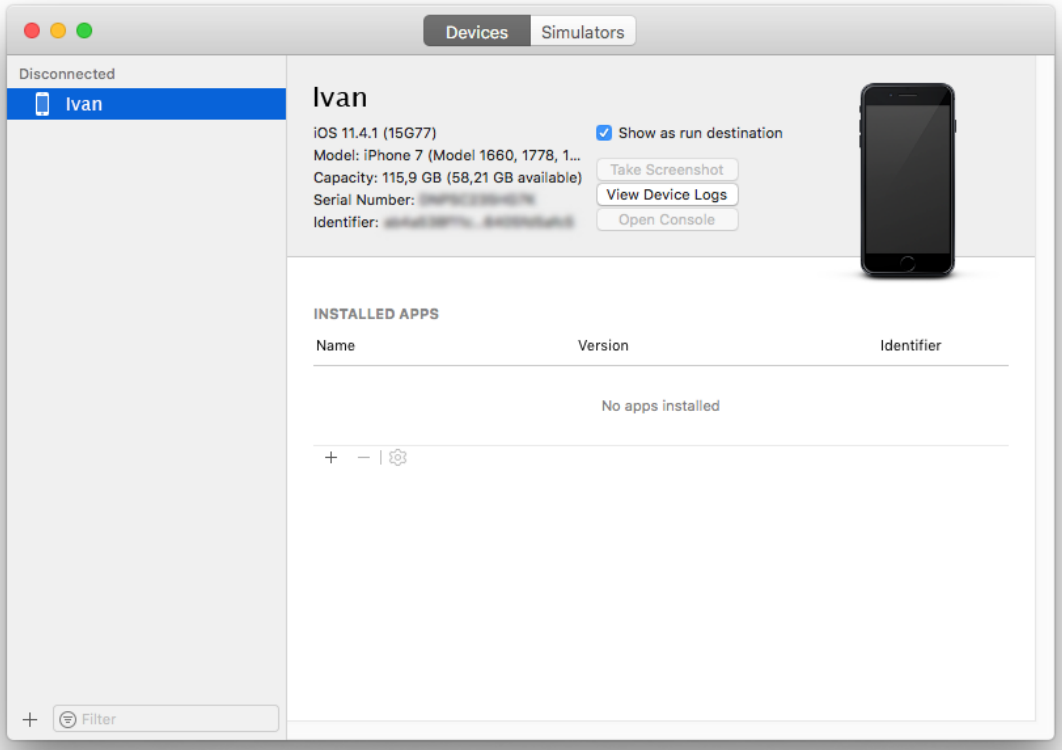


Slika 33: Odabir vrste aplikacije prilikom kreiranja novog projekta u *Xcode* sučelju (Izvor: Autor)

Na sljedećem koraku sučelje iziskuje definiranje naziva projekta, profila developera za potpisivanje aplikacije, programskog jezika, te programskog okvira. Pritom valja istaknuti da se odabire *Swift* kao programski jezik i *SceneKit* kao programski okvir.

7.1.2. Priprema uređaja za testiranje

Budući da *Xcode* simulator *iOS* uređaja ne sadrži kameru, potrebno je spojiti uređaj kako bi se *ARKit* sadržaj mogao prikazati, a samo povezivanje uređaja moguće je izvesti putem *Lightning* kabela ili bežično ukoliko su *MacOS* i *iOS* uređaji spojeni na istu mrežu.



Slika 34: *Xcode* izbornik priključenih uređaja za testiranje aplikacije

(Izvor: Autor)

7.1.3. Programiranje osnovnih elemenata aplikacije

Prije svega, budući da je na početku odabrana *Augmented Reality App* opcija, potrebno je ukloniti automatski generirani sadržaj u vidu 3D objekta za testiranje koji se nalazi u *Assets.xcassets* direktoriju pod nazivom *ship.scn* i *texture.png*.

Nadalje, u *ViewController.swift* datoteci uklonjen je višak linija koda tako da isti sadrži isključivo sljedeće linije:

```
override func viewDidLoad() {
    super.viewDidLoad()

    sceneView.delegate = self

    sceneView.showsStatistics = true
}
```

Navedene linije koda koristiti će se za potrebe aplikacije, dok je ostatak zaslužan za pokretanje *SceneKit* pogleda.

S obzirom da je cilj aplikacije prikaz izmjerene udaljenosti, potrebno je kreirati za to namijenjenu podlogu. Za kreiranje grafičkih elemenata može se koristiti *Storyboard* okruženje, međutim u ovom slučaju isto će se postići programskim putem, odnosno u kodu.

Za kreiranje prikaza deklarirana je varijabla na vrhu klase na sljedeći način:

```
var prikazMjerenja = UILabel()
```

Navedeni kod ukazuje na kreiranu instancu *UILabel* klase čiji će se raznovrsni atributi moći konfigurirati unutar *viewDidLoad()* metode.

Za kreiranje pozadine korišten je sljedeći kod u sklopu *viewDidLoad()* metode:

```
prikazMjerenja.frame = CGRect(x: 0, y: 0, width: view.frame.size.width,
height: 100)

prikazMjerenja.backgroundColor = .white
```

Prva linija koda postavlja bijelu boju pozadine za ispis mjerenja kako bi se ključni podaci istaknuli u odnosu na dinamičnu pozadinu prouzročenu otvorenim senzorom kamere na uređaju. Također, visina kreirane pozadine je definirana veličinom od 100 piksela pri vrhu zaslona.

Sljedeća stavka odnosi se na postavljanje, poravnavanje i ispis generiranog teksta. Sukladno tome, u *viewDidLoad()* metodu dodan je sljedeći kod:

```
prikazMjerenja.text = String(format: "Udaljenost: %.2f cm"
prikazMjerenja.textAlignment = .center
view.addSubview(prikazMjerenja)
```

Prva linija koda upućuje na početno stanje tekstualnog prikaza koje u aplikaciji ispisuje vrijednost od 0 cm sa dva decimalna mjesta. Druga linija koda zaslužna je za centriranje generiranog tekstualnog sadržaja na kreiranoj podlozi, dok zadnja linija dodaje *prikazMjerenja* u *view*.

S obzirom da se aplikacija bazira na postavljanju točaka u prostoru, njihovo kreiranje je nezaobilazan korak prije same implementacije koda za mjerenje udaljenosti. Nadalje, budući da je potrebno kreirati više točaka, kod se neće izvesti unutar *viewDidLoad()* metode već je potrebno kreirati novu metodu koja će se kasnije moći ponovno koristiti za pozivanje neovisno o lokaciji.

```
func novaTocka(at position: SCNVector3) -> SCNNode {
...
}
```

Iz navedenog koda se može iščitati da se uzima pozicija tipa *SCNVector3* koja vraća *SCNNode*, odnosno u ovom slučaju točku.

Za kreiranje konkretne točke koristi se sljedeći kod:

```
let tocka = SCNSphere(radius: 0.005)

let tocka = SCNNode(geometry: tocka)

node.pozicija = pozicija
```

Prva linija upućuje na kreiranje točke tipa *SCNSphere* i postavlja njen inicijalni radijus na 0.005 metara u *SceneKit* koordinatnom sustavu. Sljedeća linija koda mijenja geometriju tipa *SCNNode* u objekt koji se može mijenjati po potrebi, a *SCNNode* će se koristiti kao odgovor na poziv funkcije. Na posljetku, uzimanjem parametra pozicije iz funkcije, točka se pozicionira na mjesto gdje treba biti postavljena u skladu s dodirom zaslona uređaja.

Kako bi se točke istaknule u odnosu na pozadinu, poželjno je kreirati materijal i definirati boju materijala koji će im se dodijeliti:

```
let material = SCNMaterial()
```

U navedenom kodu, *SceneKit* dobiva informaciju da će se dodati novi materijal sa pripadajućim parametrima, odnosno kreirana je instanca *SCNMaterial()* i dodijeljena konstanti *material*.

Sljedeći korak prikazuje dodjeljivanje narančaste boje materijalu:

```
material.diffuse.contents = UIColor.orange
```

Na posljetku, potrebno je konfiguriranu rasvjetu materijala dodijeliti točki na sljedeći način:

```
material.lightingModel = .blinn


sphere.firstMaterial = material
```

Navedeni parametri definiraju na koji način će svjetla u okruženju utjecati na materijal koji je kreirarn ranije. Postavljanjem *.blinn* modela, svijetle točke scene u vidokrugu uređaja se kalkuliraju uz pomoć *Blinn-Phong*[1] formule budući da je ista dostatna za potrebe aplikacije. Nakon toga, trenutni materijal točke zamjenjuje se novo kreiranim materijalom.

[1] Standardni model za kalkuliranje nijansi rasvjete na temelju izvora svjetlosti i pozicije korisnika (McKesson, 2012.)

U konačnici, nakon kreiranja točke, potrebno je vratiti točku tipa *SCNNode* na mjesto gdje je pozvana na sljedeći način:

```
    return node
```

Ovime je okončana metoda za kreiranje točaka i vraćanje istih. Međutim, valja napomenuti kako će se točke pojaviti isključivo ukoliko se pozove metoda i potom doda u *superview*.

7.1.4. Programiranje ključnih elemenata aplikacije

Jedan od ključnih problema dotične aplikacije jest upravljanje dodirom u skladu s korisničkim inputom na osnovu stvarnog okruženja, odnosno stvaranje točaka na specifičnim lokacijama povrh stvarnog okruženja korisnika.

Prvi korak za provjeru dodira podrazumijeva kreiranje načina za prepoznavanje dodira prilikom pokretanja aplikacije što se postiže na sljedeći način:

```
    let dodirPrepoznavanje = UITapGestureRecognizer(target: self, action:
    #selector(upravljanjeDodira))
```

Ova linija koda kreira instancu *UITapGestureRecognizer()* klase i prosljeđuje dva parametra prilikom inicijalizacije, odnosno *target* i *action*. Kao prvi parametar, t*arget* predstavlja primatelja obavijesti koje se prosljeđuju na osnovu dodira, dok njegovu ulogu u ovom slučaju preuzima *ViewController* klasa. Drugi parametar, odnosno *action*, predstavlja metodu koja se poziva svaki put kada se pojavi dodir zaslona.

Za definiranje broja dodira koristi se sljedeći kod:

```
     dodirPrepoznavanje.numberOfTapsRequired = 1
```

Instanca ranije kreirane klase zahtijeva informaciju o tome koliki je broj dodira zaista potreban kako bi se aktiviralo prepoznavanje, što je u ovom slučaju jedan dodir.

Potom, potrebno je dodati upravljač dodira u *sceneView* na sljedeći način:

```
    sceneView.addGestureRecognizer(dodirPrepoznavanje)
```

Ova linija koda dodaje prepoznavanje dodira u *sceneView* gdje će se prikazivati sadržaj percipiran putem kamere uređaja i prepoznavati input korisnika putem zaslona.

Prilikom kreiranja *UITapGestureRecognizer()* klase, *upravljanjeDodira* metoda dodijeljena je *action* parametru. Slijedom toga, pri ovom koraku deklarirana je spomenuta metode na sljedeći način:

```
    @objc func upravljanjeDodira(sender: UITapGestureRecognizer) {
    ...
    }
```

Glede *@objc* oznake bitno je napomenuti kako je ista nezaobilazna u trenutnoj inačici *Swift* programskog jezika, a njezina uloga jest omogućavanje korištenja metoda u *Objective C* jeziku. Sukladno tome, *#selector* zahtijeva da je navod metode dostupan *Objetive C* jeziku, dok parametar metode daje informaciju o preciznoj lokaciji koja je dodirnuta na zaslonu.

Sljedeći korak u nastojanju prikazivanja točaka na korisnički odabranoj lokaciji na zaslonu jest prepoznavanje točne pozicije koju je korisnik označio. Kako bi se postigla navedena operacija, potrebno je dodati sljedeće linije koda unutar *upravljanjeDodira* metode:

```
    let location = sender.location(in: sceneView)

    let hitTest = sceneView.hitTest(location, types:
[ARHitTestResult.ResultType.featurePoint])

    guard let result = hitTest.last else { return }
```

Prva linija koda uzima lokaciju dodira na zaslonu u odnosu na *sceneView* i potom istu postavlja pod konstantom imena *location*.

Sljedeća linija ukazuje na *hitTest* unutar *sceneView* što podrazumijeva provjeru scene putem kamere u vidu percipiranja stvarnih objekata i površina u okruženju korisnika. Na taj način se postiže percepcija dubine, odnosno blizine i udaljenosti objekata u odnosu na kameru što u konačnici rezultira preciznim mjerama stvarnog okruženja. Također, valja naglasiti kako se *featurePoint* odnosi na tipove objekata koji su specificirani za detekciju putem kamere, a u ovom slučaju, budući da se radi o aplikaciji za mjerenje, spomenuti se odnose na ravne plohe.

Posljednja linija koda zaslužna je za preuzimanje najpreciznijeg, odnosno zadnjeg *hitTest* rezultata, nakon čega provjerava da li je *nil*. Ukoliko jest, ostatak linija koda unutar ove metode se ignorira, dok se u suprotnom dodjeljuje konstanti naziva *result*.

Ubacivanjem sljedećeg koda u *upravljanjeDodira()* metodu upotrebljuju se matrice:

```
    let transform = SCNMatrix4.init(result.worldTransform)
```

```
let vector = SCNVector3Make(transform.m41, transform.m42,
transform.m43)

let tocka = novaTocka(at: vector)
```

Budući da ranije spomenuti *hitTest* vraća *matrix_float4x4*, odnosno matricu od 4x4 *float* vrijednosti, a radnja se odvija u *SceneKit* programskom okviru, matrica se mora interpretirati u prikladnom obliku što je u ovom slučaju *SCNMatrix4*.

Potom, matrica se koristi za kreiranje *SCNVector3* vektora sa tri komponente, odnosno x, y i z vrijednostima koje predstavljaju poziciju u prostoru. Tim slijedom, *transform.m41*, *transform.m42* i *transform.m43* predstavljaju relevantne koordinatne vrijednosti triju navedenih komponenti vektora.

Na posljetku se koristi ranije kreirana metoda *novaTocka()* u kombinaciji sa parametrima lokacije deriviranih iz procesa dodira zaslona s ciljem stvaranja točke i pridruživanja konstanti imena *tocka*.

U trenutnom stadiju razvoja aplikacije evidentan je jedan propust u kodu, odnosno nove točke se kontinuirano generiraju sukladno korisničkom inputu. Budući da je kod većeg broja točaka teško procijeniti čija udaljenost se trenutno mjeri, poželjno je pojednostaviti grafičku interpretaciju trenutno korištenih točaka i time obogatiti korisničko iskustvo.

Kako bi se riješio navedeni problem, potrebno je prvo kreirati polje pri vrhu klase na sljedeći način:

```
var tocke: [SCNNode] = []
```

Navedeni kod predstavlja *SCNNode* polje budući da je to naziv tipa koji se vraća ranije kreiranom *novaTocka()* metodom. U skladu s time, kasnije će se u polje dodavati točke i provjeravati njihov ukupni broj čime se u konačnici suzbija uzastopno kreiranje većeg broja točaka.

U sljedećem koraku se koristi niz *if – else* izjava i *for* petlji kako bi se utemeljilo postoje li točke u polju:

```
if let first = spheres.first {
      ...
} else {
      ...
}
```

Navedenim kodom se prvobitno provjerava postoje li vrijednosti u polju *tocke*, te se, ukoliko ne postoje, izvršava kod unutar *else* klauzule.

Unutar *if* klauzule nalazi se sljedeći kod:

```
    tocke.append(tocka)
    prikazMjerenja.text = „Udaljenost: \(tocka.udaljenost(to: first)) cm"

    if tocke.count > 2 {

        for tocka in tocke {

            tocka.removeFromParentNode()
        }

        tocke = [tocke[2]]
    }
```

S obzirom da se kod tiče samog dodira zaslona, jasno je da će se kreirati dodatna točka. Stoga, ukoliko postoji početna točka, potrebno je derivirati udaljenost i prikazati je korisniku uz pomoć poziva metode *udaljenost()* na točki. Kako bi navedeno funkcioniralo, kasnije će se, izvan *ViewController* klase, ubaciti ekstenzija za *SCNNode*.

Nadalje, potrebno je saznati postoji li više od maksimalno dozvoljene dvije točke. Kako bi se to izvelo, koristi se *count* svojstvo polja *tocke* u kombinaciji sa *if* izjavom. Na taj se način omogućuje iteriranje kroz niz točaka u polju, kao i njihovo uklanjanje sa scene.

Na posljetku, budući da se kod nalazi unutar *if* izjave koja daje podatak da je broj točaka veći od dva, ostatak se može ukloniti iz polja na način da se veličina polja svede na maksimalno dvije točke.

U sklopu *else* klauzule, budući da je *tocke* polje prazno, potrebno je dodati točku kreiranu u trenutku poziva metode na sljedeći način:

```
    tocke.append(tocka)
```

Ovime je dodana točka u polje *tocke*, čime je samo polje spremno za sljedeći dodir zaslona, odnosno opremljeno točkama koje se trebaju prikazati na zaslonu. Kako bi se to realiziralo, potrebno je dodati sljedeći kod za iteriranje kroz točke:

```
    for sphere in spheres {

            self.sceneView.scene.rootNode.addChildNode(tocka)

    }
```

Navedeni kod predstavlja jednostavnu *for* petlju, pri čemu su točke, odnosno *SCNNode,* u ulozi djeteta *rootNode* same scene budući da je u *SceneKit* programskom okviru ovakav način dodavanja poželjan.

Slijedom prethodno navedenog, konačni kod *upravljanjeDodira()* metode izgleda ovako:

```
@objc func upravljanjeDodira(sender: UITapGestureRecognizer) {
    let location = sender.location(in: sceneView)
    let hitTest = sceneView.hitTest(location, types:
[ARHitTestResult.ResultType.featurePoint])

    guard let result = hitTest.last else { return }

    let transform = SCNMatrix4.init(result.worldTransform)
    let vector = SCNVector3Make(transform.m41, transform.m42,
transform.m43)
    let tocka = novaTocka(at: vector)

    if let first = tocke.first {
        tocke.append(tocka)
        prikazMjerenja.text = „Udaljenost: \(tocka.udaljenost(to:
first)) cm“

        if tocke.count > 2 {
            for tocka in tocke {
                tocka.removeFromParentNode()
            }

            tocke = [tocke[2]]
        }

    } else {
        tocke.append(tocka)
    }

    for tocka in tocke {
        self.sceneView.scene.rootNode.addChildNode(tocka)
    }
}
```

Ranije ubačen poziv za *udaljenost(to:)* metodu na *SCNNode*, odnosno točku, rezultira upozorenjem od strane *Xcode* sučelja. Razlog tome jest nedostatak ekstenzije za *SCNNode* klasu koja se kreira na sljedeći način:

```
extension SCNNode {
    ...
}
```

Na ovaj način se ukratko omogućuje izmjena same klase.

Nakon toga, potrebno je dodati metodu koja će odraditi izračun udaljenosti dviju točaka deklariranjem sljedeće funkcije:

```
func udaljenost(to odrediste: SCNNode) -> CGFloat {
      ...
}
```

Iz navedenog je vidljivo da postoji dodatni *SCNNode* parametar koji vraća *CGFloat* kao rezultat. Za konkretan izračun udaljenosti potrebno je dodati sljedeći kod unutar *udaljenost()* funkcije:

```
let dx = odrediste.position.x - position.x
let dy = odrediste.position.y - position.y

let dz = odrediste.position.z - position.z

let metri = sqrt(dx*dx + dy*dy + dz*dz)

return CGFloat(metri * 100)
```

Prve tri linije koda oduzimaju x, y i z *SCNNode* pozicije od koordinata točke proslijeđenih u vidu parametara. Isti se kasnije ubacuju u formulu s ciljem određivanja konačne udaljenosti.

Kako bi se odredila udaljenost dviju točaka, potrebno je implementirati formulu za udaljenost baziranu na Kartezijevom koordinatnom sustavu i primijeniti je u trodimenzionalnom prostoru.

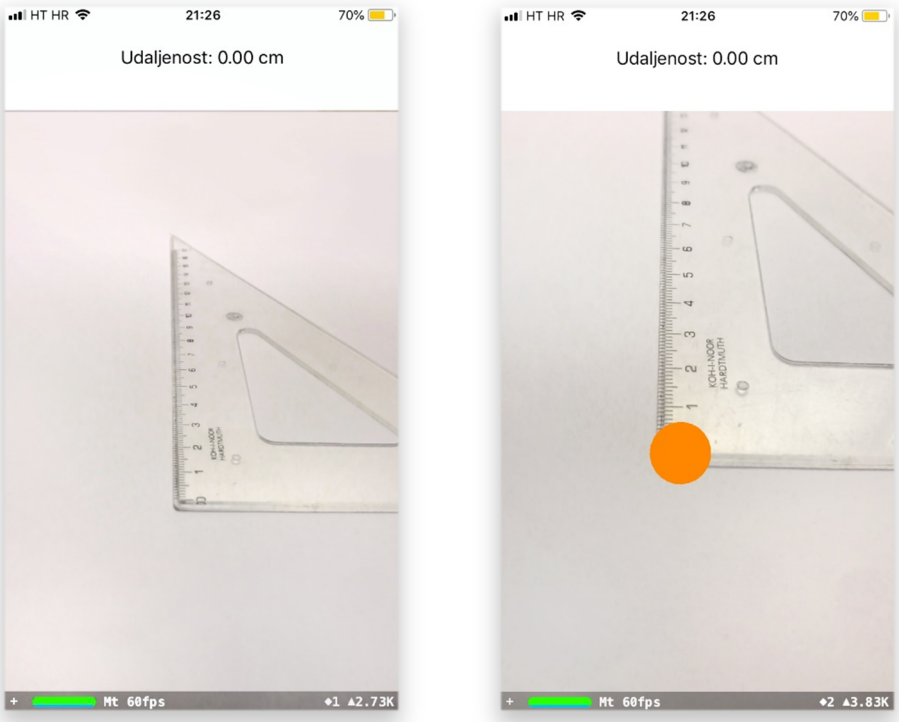
Na posljetku, vrijednost se multiplicira kako bi se konačna udaljenost na zaslonu ispisala u centimetrima.

7.1.5. Testiranje kreirane aplikacije

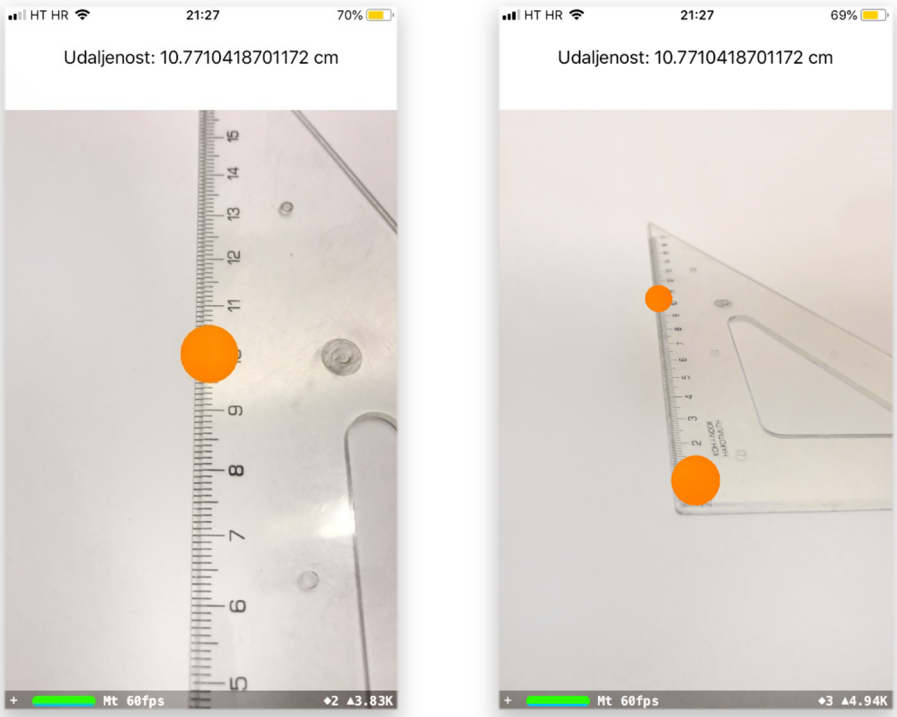
Na temelju kombinacije svih ranije navedenih instanci koda može se pokrenuti finalna verzija aplikacije. Po pokretanju može se zamijetiti kraći period stanke prilikom kojeg se odvijaju standardne *ARKit* kalkulacije kao što su detekcija horizontalnih i vertikalnih ploha, prepoznavanje dubine i udaljenosti u odnosu na kameru uređaja, te kompenziranje rasvjete ovisno o okruženju.

Nakon učitavanja potrebnih elemenata, aplikacija je spremna za postavljanje točaka u prostoru, odnosno primanje korisničkog inputa. Iako je točke moguće postavljati gotovo bilogdje u vidokrugu kamere, preporučljivo je korištenje na udaljenosti od 10 cm do 300 cm za preciznije rezultate. Također, valja napomenuti kako se preciznije kalkulacije dobivaju ukoliko se korisnik približi željenoj točki u prostoru prilikom postavljanja, neovisno o tome da li se početna i

krajnja točka istovremeno mogu vidjeti na zaslonu. Razlog tome jest praćenje i mapiranje stvarnog okruženja u odnosu na točku iz čega proizlazi zaključak da se iste postavljaju kao markeri povrh okruženja, odnosno utjelovljuju definiciju proširivanja stvarnosti korisnika.



Slika 35: Početni zaslon prilikom pokretanja aplikacije i postavljanje početne točke (Izvor: Autor)



Slika 36: Odabir krajnje točke mjerenja i prikaz obiju točaka vezanih za objekt u prostoru (Izvor: Autor)

Iz priloženih slika može se zaključiti kako prilikom pokretanja aplikacije i inicijalizacije koda za prepoznavanje ploha i objekata udaljenost ostaje nepromijenjena, odnosno iznosi 0,00.

Po završetku procesa prepoznavanja podloge, korisnik je u mogućnosti postaviti početnu točku na željenu lokaciju, a nakon postavljanja krajnje točke mjerenja može se primijetiti kako je izmjerena udaljenost poprilično precizna uzme li se u obzir veličina točaka.

Na posljednjoj slici moguće je prepoznati prisutnost proširene stvarnosti budući da se zadane točke vezuju za definiranu poziciju na podlozi i ostaju zapisane u prostoru sve do ponovnog zadavanja nove početne točke za mjerenje, neovisno o promjeni pozicije uređaja.

Slijedom navedenog, evidentno je isticanje nekih od ključnih karakteristika, tj. pogodnosti *Apple ARKit* biblioteke.

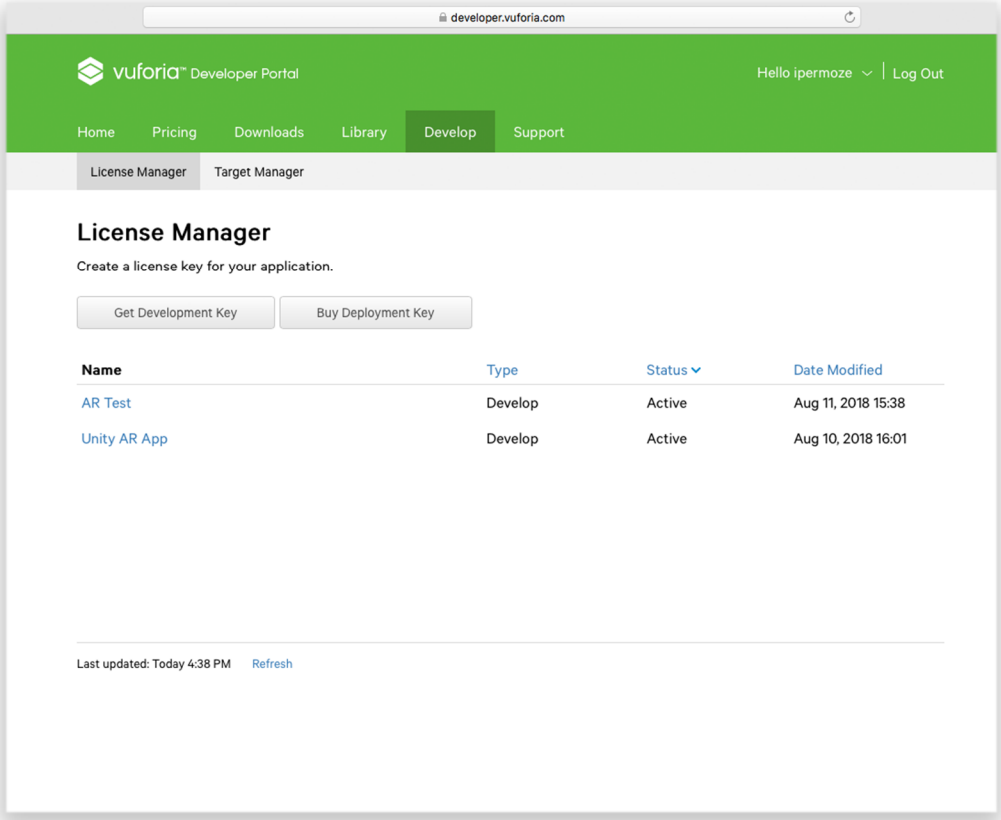
7.2. Aplikacija za prikaz trodimenzionalnog modela na temelju markera

U ovom poglavlju obrađen je proces izrade aplikacije za implementaciju proširene stvarnosti na *iOS* platformi uz pomoć *Unity 3D* softvera za razvoj igara, kao i *Vuforia* dodatka za spomenuti softver. Ideja aplikacije temelji se na prikazu virtualnog objekta povrh preddefiniranih markera uz mogućnost kretanja kroz prostor na temelju virtualnih kontrola vezanih za sami objekt.

7.2.1. Priprema softverske podloge

Prije svega, inicirano je preuzimanje ažurirane inačice *Unity 3D* softvera, kao i dodatnih paketa za željenu platformu za razvoj, odnosno *iOS* platformu.

Kako bi se kamera uređaja mogla koristiti za percipiranje okoline, preuzeta je i ažurirana inačica *Vuforia* dodatka za *Unity 3D* softver sa službene Web stranice koja iziskuje registraciju korisnika za potrebe generiranja ključeva za pojedine markere koji će se kasnije definirati.



Slika 37: *Vuforia Developer Portal*

(Izvor: Autor)

Putem *Develop* poveznice na službenoj *Vuforia* Web stranici dodijeljena je licenca, odnosno ključ koji će se kasnije vezati uz marker.

Nakon toga, putem *Target Manager* poveznice kreirana je nova baza podataka prikladnog naziva kako bi se u istu mogla učitati proizvoljno odabrana slika koja će preuzeti ulogu markera prilikom korištenja aplikacije.

Za potrebe demonstracije navedenog, odabrana je širina mete koja iznosi pet jedinica, dok su u *AR Test* bazu pohranjeni markeri imena *Unity_AR_Test_Marker1* i *Unity_AR_Test_Marker2* koji predstavljaju *QR* kod (Slika 8.) i logotip Sveučilišta Jurja Dobrile u Puli kao željene markere za kasniju upotrebu.

7.2.2. Kreiranje trodimenzionalnog modela za virtualni prikaz

Za svrhe prikazivanja funkcionalnosti aplikacije izrađen je trodimenzionalni model Unipu logotipa na osnovu materijala dostupnog na stranicama Sveučilišta Jurja Dobrile u Puli.

Model logotipa izrađen je pomoću *Cinema 4D R19* programa za trodimenzionalne animacije. Nakon izrade svakog zasebnog segmenta i dodavanja tekstura u odgovarajućim bojama, trodimenzionalni model je eksportiran u obliku *Autodesk .fbx* datoteke budući da *Unity 3D* sustav podržava isključivo *.fbx*, *.dae*, *3ds*, *.dxf*, *.obj*, i *.skp* oblike datoteka sa pratećim teksturama. S obzirom na limitacije *Cinema 4D* programa, eksportiranje u formatima koji nisu srodni *.c4d* ekstenziji, inače tvorničkom formatu ovog programa, rezultira u gubitku boja i tekstura trodimenzionalnih modela. Međutim, iznimka ovog pravila jest *.fbx* oblik datoteka, stoga je isti upotrijebljen prilikom importiranja modela u *Unity 3D* softver.

Također, izrađene su dvije verzije logotipa, odnosno statična i dinamična. Dok statična verzija sadrži samo komponente i teksture, animirana verzija pored navedenog sadrži i samu animaciju. Zahvaljujući ažuriranoj verziji *Unity 3D* softvera, *.fbx* datoteke s popratnim animacijama više ne predstavljaju izazov po pitanju reprodukcije animiranog sadržaja objekta, stoga je isti iskorišten za prikazivanje u sklopu konačne verzije izrađene aplikacije.

Slika 38: Proces izrade trodimenzionalnog modela Unipu logotipa

(Izvor: Autor)

7.2.3. Kreiranje aplikacije i implementacija modela

Prvi korak pri kreiranju aplikacije podrazumijeva kreiranje novog projekta unutar *Unity 3D* softvera, kao i definiranje željene platforme za razvoj unutar *Build Settins* postavki.

Sljedeća važna stavka jest ubacivanje baze podataka za ranije definiranu metu, odnosno markere, te importiranje trodimenzionalnog modela u *.fbx* formatu nakon čega je poželjna pohrana kreiranog projekta.

Kako bi praćenje scene, odnosno stvarnog okruženja funkcioniralo, potrebno je kreirati novi *Vuforia* element imena *ARCamera* i ukloniti postojeću kameru sa scene. Budući da najnovija verzija *Unity 3D* softvera sadrži kompletan niz *Vuforia* dodataka, iste nije potrebno preuzimati sa službene Web stranice, već ih je moguće naći u padajućem izborniku dostupnom putem desne tipke miša.

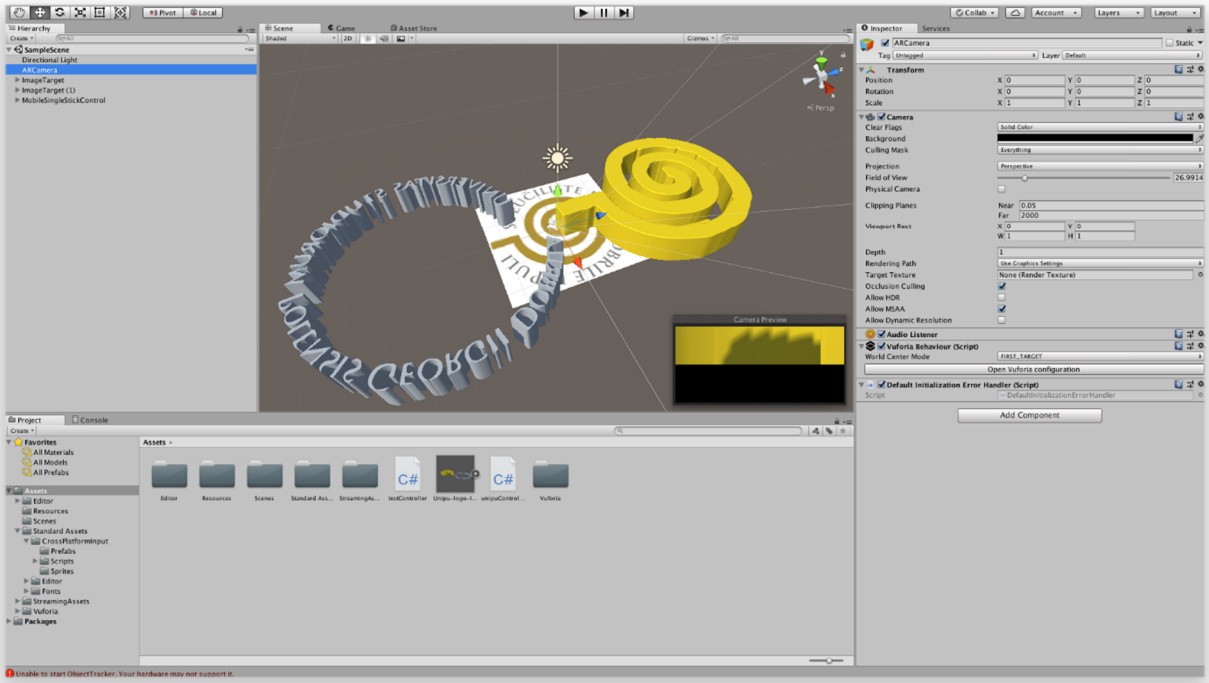
Nakon kreiranja kamere za proširenu stvarnost, kreirana je slikovna meta, odnosno marker za prikazivanje virtualnog objekta za čije potrebe će se koristiti element imena *ImageTarget* koji se također može naći u sklopu *Vuforia* padajućeg izbornika. Budući da će se u svrhu prikazivanja sadržaja koristiti dva različita markera, za svaki je kreirana zasebna *ImageTarget* instanca.

Odabirom *ARCamera* elementa u sklopu scene, pod *Inspector* karticom, može se zamijetiti *Vuforia Behaviour* stavka koja služi za konfiguriranje skripte koja definira ponašanje elemenata prilikom upotrebe proširene stvarnosti.

App License Key rubrika ključna je prilikom izrade aplikacije budući da sadrži ranije pohranjeni ključ za povezivanje markera sa *Vuforia* dodatkom, odnosno povezuje *ImageTarget* slike sa operacijom prikazivanja virtualnog sadržaja prilikom pokretanja same aplikacije. Pritom, valja napomenuti kako bazu podataka za željene markere nije potrebno zasebno učitavati na istoimenoj poveznici unutar *Vuforia* postavki budući da se ista učitala prilikom inicijalnog importiranja baze na samom početku.

Kako bi se trodimenzionalni objekt mogao prikazati, potrebno ga je učitati unutar *ImageTarget* elementa u ulozi „djeteta“. U skladu s načinom na koji *Unity 3D* interpretira importirani objekt, isti je, pod *Transform* komponentom, potrebno rotirati za 180 stupnjeva kako bi se mogao pravilno prikazati povrh markera. Nadalje, pozicija objekta je pomaknuta za - 0,7 jedinica na x osi, dok su dimenzije sa 1 smanjene na 0.3 kako bi veličina objekta odgovarala veličini samog markera.

Za potrebe demonstracije kretanja objekta u prostoru implementiran je kontroler iz standardne *Unity 3D* biblioteke pod imenom *MobileSingleStickController* koji se može pronaći unutar *CrossPlatformInput* direktorija. Iako spomenuti kontroler sadrži i *Jump* komopentu, ista je uklonjena budući da planirani raspon kretanja objekta podrazumijeva isključivo horizontalne koordinate. Veličina kontrolera postavljena je na dvije jedinice, dok je raspon kretanja ograničen na 50. Nakon toga, trodimenzionalnom je objektu dodijeljena *Rigidbody* komponenta i uklonjena *Use Gravity* opcija.



Slika 39: Proces izrade aplikacije u *Unity 3D* softveru

(Izrvor: Autor)

7.2.4. Kreiranje skripte za kretanje objekta u prostoru

Nakon dodjeljivanja *Rigidbody* komponente, objektu je također dodijeljena novo kreirana *C#* skripta koja definira uvjete interakcije postojećih elemenata.

Kako bi se konačna aplikacija mogla pokrenuti, prilikom iniciranja skripte unutar *Visual Studio* aplikacije potrebno je ustvrditi da se koriste sljedeće biblioteke: *System.Collections*, *System.Collections.Generic*, *UnityEngine*, te *UnityStandardAssets.CrossPlatformInput*.

Potom je definirana privatna *Rigidbody* varijabla koja je izjednačena sa *GetComponent* unutar *Start()* funkcije kako bi se izbjeglo korištenje *GetComponent* unutar *Update()* funkcije koja se poziva pri svakom osvježenju slike.

Unutar *Update()* funkcije kreirane su *float x* i *float y* kako bi se izjednačile sa *CrossPlatformInputManager.GetAxis* funkcijama za horizontalno i vertikalno pomicanje objekta pomoću kontrolera.

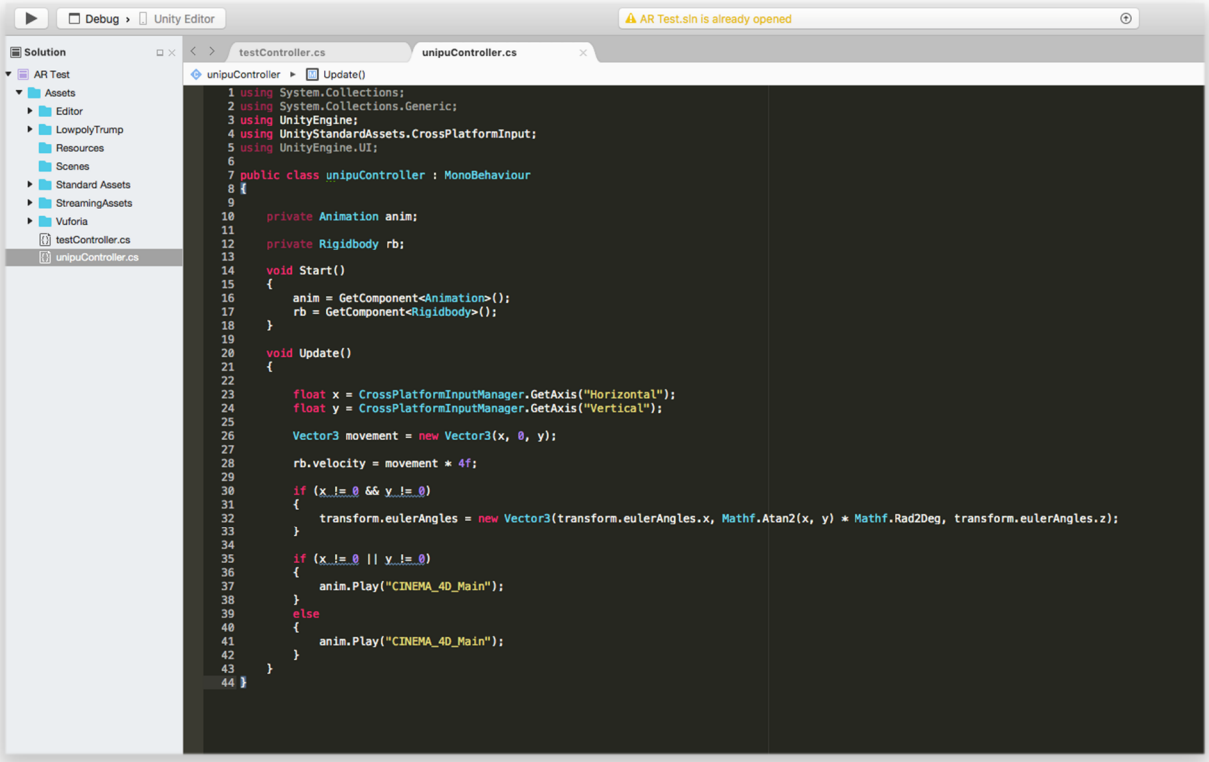
Kreiranjem *Vector3 movement* vektora postiže se dvodimenzionalno kretanje kontrolera na osnovu uobičajenih trodimenzionalnih naredbi time što su postavljene tek dvije od tri koordinate, odnosno y koordinata je izjednačena s nulom, dok je z koordinati dodijeljen naziv

y. Ovakav način definiranja koordinata opravdan je činjenicom da će se objekt kretati isključivo na horizontalnoj površini.

Sljedeći potreban parametar odnosi se na brzinu kretanja objekta u prostoru. Tim slijedom, *rb.velocity = movement * 4f* određuje brzinu kretanja od četiri jedinice.

Budući da je zamišljeno da se objekt okreće u skladu sa smjerom kretanja, potrebno je stvoriti uvjet pri kojem se ispituje jesu li *x* i *y* varijable jednake nuli. Uzme li se u obzir da je rotacija potrebna samo na y osi, x i z će se izjednačiti sa trenutnim x i z vrijednostima, dok će se za y koristiti rotacija u stupnjevima zahvaljujući *Mathf.Atan2* funkciji. S obzirom da spomenuta funkcija vraća rezultat u radijanima, na posljetku će se pomnožiti sa *Mathf.rad2deg* kako bi se dobio konačan rezultat u stupnjevima (Unity 3D, n.d.).

Nakon definiranja optimalnog načina kretanja, preostaje pridruživanje animacija koje će se pokretati na osnovu definiranih uvjeta u skripti. Budući da je kreirana samo jedna animacija za svrhe demonstracije funkcionalnosti aplikacije, ista će biti primijenjena za oba uvjeta.



Slika 40: Prikaz procesa programiranja *unipuController.cs* skripte za upravljanje objekta u prostoru unutar *Visual Studio* aplikacije

(Izvor: Autor)

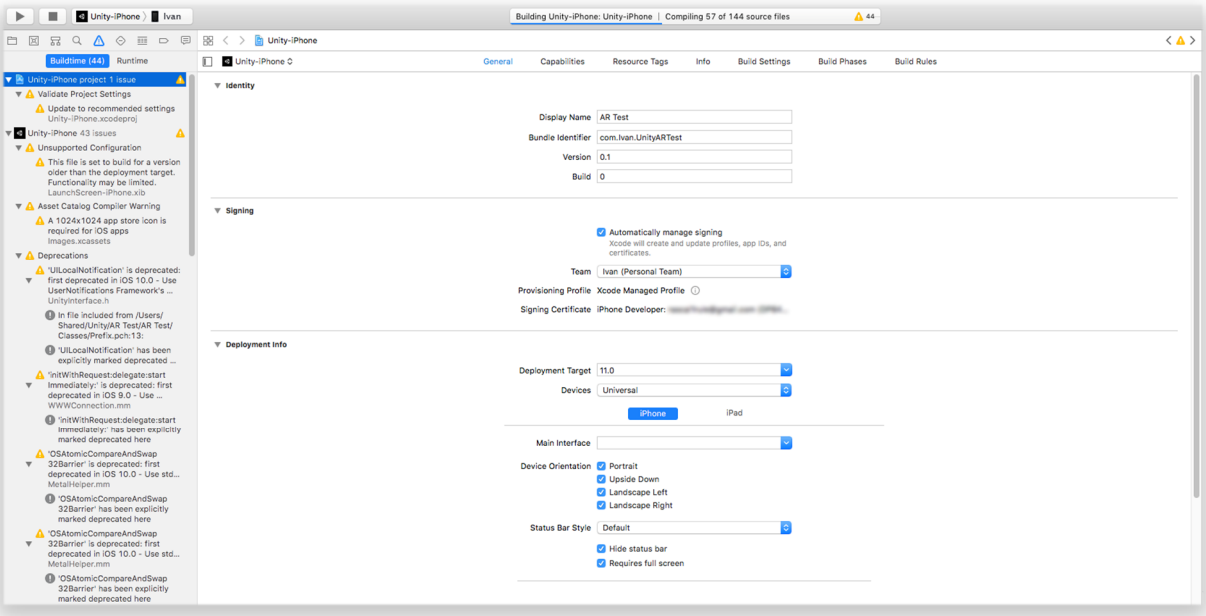
Kako bi se animacije pravilno izvodile, na samom je objektu pod *Rig* karticom podešen tip animacije na *Legacy*, a potom je pod *Animation* karticom označena *Import Animation* opcija nakon čega su potvrđene zadane promjene.

Pod objektom na sceni veličina animacije unutar *Animation* komponente podešena je na dvije jedinice, potom su odabrane importirane animacije, te je na posljetku kreirana referenca za spomenutu promjenu unutar pripadajuće funkcije u skripti. Na taj način je omogućeno automatsko pokretanje preddefinirane animacije na osnovu trenutne operacije, odnosno načina kretanja objekta.

Posljednji ključan detalj vrijedan spomena nalazi se pod *Vuforia Behaviour* komponentom unutar *ARCamera* elementa. Kako bi se definirala precizna orijentacija virtualnog objekta, potrebno je *World Center Mode* opciju podesiti na *FIRST_TARGET*. Na taj se način ostatak virtualiziranja odvija na temelju prvog percipiranog markera, u odnosu na klasično orijentiranje putem pozicije uređaja u prostoru koje ne daje konzistentne rezultate.

Za pokretanje kreirane aplikacije odabrana je i pokrenuta *Build Settings* opcija, potom su dodane otvorene scene putem *Add Open Scenes*, te je osigurana adekvatna platforma za konačan output svih elemenata. Budući da se output vrši za *iOS* platformu, uvjet za uspješno izvršavanje ove operacije jest posjedovanje *Xcode* aplikacije.

Nakon pokretanja putem *Build And Run* opcije, budući da u kodu nisu evidentirane greške, otvorena je *Xcode* inačica gdje je naknadno specificiran vlastiti developerski certifikat za potpisivanje aplikacije kako bi se ista u konačnici uspješno pokrenula na samom uređaju.



Slika 41: Proces pokretanja kreirane aplikacije putem *Xcode* sučelja

(Izvor: Autor)

Na temelju kreirane aplikacije može se zaključiti kako *Vuforia* predstavlja korisnu alternativu u odnosu na *Apple ARKit* biblioteku budući da se izrađeni projekti mogu eksportirati na niz različitih platformi bez potrebe za mijenjanjem definiranih parametara i pozicije elemenata na sceni.

Iako je detekcija definiranih markera precizna, evidentna je činjenica da za nesmetanu operaciju i dosljedne rezultate *Vuforia* iziskuje što bolju rasvjetu kako bi se smanjila količina šuma u inputu putem kamere uređaja i time u konačnici osigurala oštra percepcija stvarnog okruženja.

Slika 42: Testiranje funkcionalnosti izrađene aplikacije

(Izvor: Autor)

7.3. Aplikacija za prepoznavanje predmeta u okolini

U odnosu na prethodno predstavljene načine implementacije proširene stvarnosti, u sljedećoj aplikaciji primijenjeni su klasični elementi *ARKit* i *SceneKit* biblioteka u kombinaciji sa *Core ML*, odnosno *Vision* bibliotekom.

Dok su *ARKit* i *SceneKit* biblioteke zaslužne za percepciju površina i pohranu proizvoljno definiranih parametara povrh stvarnog okruženja korisnika, *Vision* biblioteka uvodi mogućnost upotrebe potencijala računalnog učenja (engl. Machine learning) i računalne vizije (engl. Computer vision) kako bi se proizvoljno odabranim točkama u prostoru dodijelila tekstualna definicija, tj. opis na temelju kategorije kojoj predmet pripada. Procjena kategorije, odnosno vrste predmeta odrađuje se uz pomoć preddefiniranih modela računalnog učenja kao što su *Inceptionv3*, *Resnet50*, *MobileNet*, *SqueezeNet*, *GoogLeNetPlaces* i *VGG16*, inače dostupnih u sklopu *Apple Developer* dokumentacije, te vlastito izrađenog modela na hrvatskom jeziku.

7.3.1. Kreiranje novog projekta

Nalik ranije primijenjenom načinu, ova aplikacija se također bazira na *Augmented Reality App* šabloni, odnosno koristi standardne elemente aplikacije proširene stvarnosti koji će se naknadno nadograditi.

Budući da će se za prikaz točaka i tekstualnog opisa predmeta koristiti trodimenzionalni objekti, kao programski okvir odabran je *SceneKit*, dok *Swift* ostaje definiran kao preferirani programski jezik.

Nakon kreiranja projekta, prije same obrade programskog dijela, definirano je nekoliko grafičkih elemenata unutar *Main.Storyboard* datoteke koja se odnosi na grafički prikaz aplikacije na uređaju.

Iz biblioteke objekata, dostupne s desne strane *Xcode* sučelja, implementiran je *ARKit Scene View* element, inače zaslužan za prikaz elemenata proširene stvarnosti, čiji je raspon definiran na način da obuhvaća vidno polje zaslona u cijelosti putem *Constraints* opcije unutar *Size Inspector* kartice *Xcode* sučelja.

Potom, iz biblioteke objekata dodana je *Button* komponenta čiji parametri uključuju *Alpha* vrijednost od jedne jedinice, transparentnu pozadinu, te horizontalno i vertikalno centriranu *Aligmnent* opciju unutar *Attributes Inspector* kartice. Razlog navedenih parametara jest

činjenica da će se *Button* element koristiti isključivo kao reprezentacija centralne točke zaslona budući da će se u sklopu kasnijeg koda centralna točka zaslona definirati kao lokacija na kojoj će se pojavljivati virtualni sadržaj na osnovu korisničkog inputa.

7.3.2. Programiranje aplikacije

Kako bi se kod u nastavku uspješno izvršio potrebno je osigurati da se koriste sljedeće biblioteke:

```
import UIKit
import SceneKit
import ARKit
import Vision
```

Nadalje, u sklopu *ViewController* klase nakon automatski definiranog *@IBOutlet* atributa za potrebe *ARSCNView* komponente, dodani su sljedeći parametri:

```
let textDubina : Float = 0.01

var MLProcjena : String = "…"
```

Pomoću *let* prefiksa definirana je konstanta imena *textDubina* čime je u metrima određena dubina trodimenzionalnog teksta koji će se kasnije pojavljivati u skladu s inputom od strane korisnika.

Varijabla imena *MLProcjena* definirana je kako bi sadržavala *Core ML* predviđanje, odnosno procjenu kategorije kojoj pripada predmet percipiran putem kamere uređaja.

Nezaobilazni parametri za *Core ML* funkcionalnost uključuju:

```
var vizijaZahtjev = [VNRequest]()
let dispatchQueueML = DispatchQueue(label: "DispatchQueueML")
```

VNRequest predstavlja apstraktnu klasu čija je uloga procesiranje zahtjeva za analizu slike, a njezine parametre nasljeđuju ostali zahtjevi temeljeni na *Vision* biblioteci.

DispatchQueue se odnosi na upravljanje procesa izvršavanja radnih zadataka koji se obrađuju redosljedom definiranim od strane sustava.

Nadalje, unutar standardne *viewDidLoad()* funkcije, pored automatski generiranih elemenata, valja istaknuti sljedeći dio koda za prepoznavanje dodira:

```
let dodir = UITapGestureRecognizer(target: self, action:
#selector(self.upravljanjeDodirom(dodirPercepcija:)))
```

```
        view.addGestureRecognizer(dodir)
```

Konstanta imena *dodir* sadrži podklasu *UIGestureRecognizer* klase koja je zaslužna za prepoznavanje određenog broja dodira čiji *target* i *action* parametri služe za pokretanje alociranog objekta, odnosno *self* ekspresija upućuje na instancu klase u kojoj se izvršava. Ostatak *#selector* sintakse odnosi se na *upravljanjeDodirom(dodirPercepcija:)* funkciju koja će se kasnije definirati u sklopu *@objc* dijela koda, dok se *view.addGestureRecognizer* odnosi na dodjeljivanje funkcionalnosti prepoznavanja dodira varijabli *view*, odnosno pogledu.

Za potrebe postavljanja parametara *Core ML* modela koristi se kod u nastavku:

```
guard let odabirModela = try? VNCoreMLModel(for: VGG16().model) else
{
            fatalError("Nije moguće učitati model.")
  }
```

Općenita primjena izjave *guard* se odnosi na prijenos kontrole programa ukoliko se jedan ili više uvjeta ne ispune. U ovom slučaju, ukoliko se korišteni *VGG16* model računalnog učenja ne učita, odnosno nije prisutan unutar direktorija u kojem se nalazi *ViewController* datoteka, *Xcode* će ispisati „Nije moguće učitati model.“ grešku. Tim slijedom, u slučaju promjene korištenog modela računalnog učenja, potrebno je pripadajući naziv upisati u sklopu *for* uvjeta i sam model pohraniti u prikladnom direktoriju. Također, valja istaknuti kako će se konačna veličina aplikacije povećati sukladno modelu računalnog učenja koji se trenutno koristi, odnosno *.mlmodel* datoteka je zaslužna za povećanu zapreminu potrebnu za pohranu cjelokupne aplikacije na uređaju.

Sljedeći segment koda se odnosi na postavljanje *Core ML* zahtjeva za računalnu viziju:

```
      let klasifikacijaZahtjev = VNCoreMLRequest(model: odabirModela,
completionHandler: upravljacIzvrseneKlasifikacije)
      klasifikacijaZahtjev.imageCropAndScaleOption =
VNImageCropAndScaleOption.centerCrop
      vizijaZahtjev = [klasifikacijaZahtjev]

      loopCoreMLUpdate()
```

Konstanta *klasifikacijaZahtjev* sadrži *VNCoreMLRequest* klasu koja predstavlja zahtjev za analizu slike uz pomoć *Core ML* modela. Kao *model* parametar u sklopu klase postavljena je ranije definirana *odabirModela* konstanta, dok *completionHandler* ulogu preuzima funkcija *upravljacIzvrseneKlasifikacije* koja će se kasnije definirati.

Varijabla *imageCropAndScaleOption* odnosi se na parametar koji *Vision* algoritmu daje informaciju kako postupiti sa veličinom slike, odnosno inputa. Tim slijedom,

VNImageCropAndScaleOption.centerCrop upućuje na korištenje slike koja se nalazi u centru vidnog polja i potom rasteže do prikladne veličine.

Na posljetku sintakse, inicijalno definirana varijabla *vizijaZahtjev* izjednačena je sa konstantom *klasifikacijaZahtjev* kako bi se ista proslijedila *VNRequest* klasi za upravljanje zahtjevima temeljenim na analizi slike, a potom je uz pomoć *loopCoreMLUpdate()* funkcije osigurano kontinuirano osvježenje *Core ML* kalkulacije.

Sljedeći segment sintakse koji valja istaknuti uključuje *@objc* atribut koji upućuje na dio koda u *Objective C* jeziku:

```
@objc func upravljanjeDodirom(dodirPercepcija:
UITapGestureRecognizer) {

      let zaslonSredina : CGPoint = CGPoint(x:
self.sceneView.bounds.midX, y: self.sceneView.bounds.midY)

      let arHitTestResults : [ARHitTestResult] =
sceneView.hitTest(zaslonSredina, types: [.featurePoint])


      if let najbliziRezultat = arHitTestResults.first {

            let pretvorba : matrix_float4x4 =
      najbliziRezultat.worldTransform
            let koordinate : SCNVector3 =
      SCNVector3Make(pretvorba.columns.3.x, pretvorba.columns.3.y,
      pretvorba.columns.3.z)

            let tocka : SCNNode = novi3DText(MLProcjena)
            sceneView.scene.rootNode.addChildNode(tocka)
            tocka.position = koordinate
     }
}
```

Funkcija *upravljanjeDodirom* zaslužna je za povezivanje interakcije sa zaslonom putem *UITapGestureRecognizer* podklase.

Kako bi se sredina zaslona pohranila, uz konstantu *zaslonSredina* korištena je struktura *CGPoint* koja sadrži točku u dvodimenzionalnom koordinatnom sustavu. Slijedom toga, x i y koordinate definirane su putem *bounds.midX* i *bounds.midY* varijabli koje u prijevodu centriraju točku na sredini zaslona.

ARHitTestResult klasa, inače komponenta *ARKit* biblioteke, zaslužna je za prikupljanje informacija temeljenih na analizi okoline percipirane putem kamere uređaja. Tim putem se percipirano okruženje povezuje sa sredinom zaslona gdje se stvara točka u prostoru temeljena

na *.featurePoint* statičnoj varijabli koja se vezuje za površinu zahvaljujući *ARKit point cloud* reprezentaciji stvarnog okruženja.

Potom, ukoliko najbliži rezultat odabrane točke odgovara najbližoj točki *ARKit point cloud* reprezentacije stvarnog okruženja na temelju koordinata *worldTransform* varijable i *SCNVector3* vektora, inicira se kreiranje trodimenzionalnog teksta putem *tocka* konstante temeljene na *MLProcjena* varijabli koja daje podatak o najnovijoj *Core ML* procjeni kategorije kojoj promatrani predmet pripada.

Funkcija *novi3DText* definirana je izvan obima *upravljanjeDodirom(dodirPercepcija:)* funkcije, a njena sintaksa sadrži sljedeće:

```
    func novi3DText(_ text : String) -> SCNNode {

            let natpisOgranicenje = SCNBillboardConstraint()
            natpisOgranicenje.freeAxes = SCNBillboardAxis.Y

            let Text3D = SCNText(string: text, extrusionDepth:
    CGFloat(textDubina))
            var font = UIFont(name: "EuphemiaUCAS", size: 0.15)
            font = font?.saOsobinama(osobine: .traitBold)
            Text3D.font = font
            Text3D.alignmentMode = kCAAlignmentCenter
            Text3D.firstMaterial?.diffuse.contents = UIColor.orange
            Text3D.firstMaterial?.specular.contents = UIColor.white
            Text3D.firstMaterial?.isDoubleSided = true

            Text3D.chamferRadius = CGFloat(textDubina)

            let (minBound, maxBound) = Text3D.boundingBox
            let Text3DNode = SCNNode(geometry: Text3D)

            Text3DNode.pivot = SCNMatrix4MakeTranslation( (maxBound.x -
    minBound.x)/2, minBound.y, textDubina/2)
            Text3DNode.scale = SCNVector3Make(0.2, 0.2, 0.2)

            let kugla = SCNSphere(radius: 0.005)
            kugla.firstMaterial?.diffuse.contents = UIColor.white
            let kuglaNode = SCNNode(geometry: kugla)

            let Text3DNodeRoditelj = SCNNode()
            Text3DNodeRoditelj.addChildNode(Text3DNode)
            Text3DNodeRoditelj.addChildNode(kuglaNode)
            Text3DNodeRoditelj.constraints = [natpisOgranicenje]

            return Text3DNodeRoditelj
        }
```

Sama *novi3DText* funkcija definirana je kao tekstualna *String* struktura koja vraća *SCNNode* klasu, odnosno strukturalni element percipirane scene koji predstavlja trodimenzionalne

koordinate percipirane scene uz mogućnost dodavanja komponenti kao što su geometrija, rasvjeta, kamere i slično.

SCNBillBoardConstraint() klasa odnosi se na ograničenje čvora, tj. usmjerenje prema trenutno definiranoj kameri na način da se z koordinata uskladi sa točkom gledišta na osnovu koje se generira trenutna scena. U ovom slučaju, na temelju *SCNBillboardAxis.Y* strukture, y koordinata čvora postaje trajno paralelna sa zaslonom uređaja.

Konstanta *Text3D* sadrži *SCNText* varijablu koja predstavlja geometrijski element temeljen na nizu tekstualnih znakova i prethodno definirane dubine, odnosno treće dimenzije objekta koji će se prikazivati u prostoru.

Klasa *UIFont* zaslužna je za definiranje vrste i veličine fonta koji će se koristiti prilikom prikazivanja objekta, a najprikladnije parametre u ovom slučaju predstavljaju *EuphemiaUCAS* font i veličina od 0.15 jedinica. Kako bi se manjak poligonalnih elemenata u objektu kompenzirao, fontu je pripisana osobina u vidu statične varijable *traitBold* koja predstavlja zadebljanje prikazanog teksta.

Nadalje, izjednačavanjem *alignmentMode* varijable sa *kCAAlignmentCenter* konstantom, postignuto je vizualno poravnanje tekstualnog elementa povrh centralne točke.

Korištenjem *firstMaterial* metode upotrjebljene su *diffuse*, *specular*, te *isDoubleSided* varijable. Raspršenost svjetla regulirana je od strane *diffuse* varijable, a pripadajućem objektu, odnosno tekstu dodijeljena je narančasta boja. Točki, odnosno kugli povrh koje će se prikazivati tekstualni objekt, dodijeljena je bijela boja uz *specular* varijablu zaslužnu za upravljanje refleksijama. Budući da *specular* varijabla bez definiranih parametara predstavlja crnu boju, u ovom slučaju njena prisutnost rezultira u prividno matiranoj površini objekta. Potom, uz pomoć *isDoubleSided* varijable osigurano je da se odabrani materijal procesira korištenjem *SceneKit* biblioteke, odnosno prikaže sa obije strane objekta.

Željeni obrub objekta definiran je putem *chamferRadius* varijable koja preuzima prethodno zadane *textDubina* parametre.

Dodjeljivanjem *pivot* varijable *Text3DNode* konstanti omogućeno je definiranje pozicije, rotacije i veličine čvora putem *SCNMatrix4MakeTranslation* funkcije koja uzima minimalne i maksimalne parametre okvira unutar kojeg se tekst nalazi.

Konstanta *kugla* definirana je pomoću *SCNSphere* klase koja predstavlja geometriju kugle, a njezin radijus postavljen je na 0.005 jedinica.

Na posljetku, konstanti *Text3DNodeRoditelj* pridružene su *addChildNode* funkcije koje sadrže *Text3DNode* i *kuglaNode* konstante, a kao ograničenje preuzeti su parametri ranije definirane *natpisOgraničenje* konstante.

Kako bi se uspješno prikazala klasifikacija, odnosno procjena percipiranih objekata, korištena je sljedeća sintaksa:

```
func upravljacIzvrseneKlasifikacije (zahtjev: VNRequest, greska:
Error?) {

      if greska != nil {
            print("Greška: " + (greska?.localizedDescription)!)
            return
        }
        guard let zapazanja = zahtjev.results else {
            print("Nema rezultata")
            return
        }
```

Navedeni dio sintakse odnosi se na ispis grešaka. Ukoliko varijabla *greska* nije jednaka nuli, odnosno postoji, ispisati će se tekstualna reprezentacija greške uz pomoć *localizedDescription* varijable. U suprotnom, ukoliko ne postoje percipirani rezultati, ispisati će se "Nema rezultata".

```
let klasifikacije = zapazanja[0...1]

      .flatMap({ $0 as? VNClassificationObservation })
      .map({ "\($0.identifier) \(String(format:"- %.2f",
$0.confidence))" })
      .joined(separator: "\n")


DispatchQueue.main.async {

      print(klasifikacije)
      print("--")


      var imeObjekta:String = "…"
      imeObjekta = klasifikacije.components(separatedBy: "-")[0]
      imeObjekta = imeObjekta.components(separatedBy: ",")[0]
      self.MLProcjena = imeObjekta

      }
}
```

Iz ostatka sintakse se može zamijetiti kako konstanta *klasifikacije* vraća konstantu *zapazanja*, odnosno dva najpreciznija rezultata procjene.

.flatMap funkcija zaslužna je za vraćanje polja koje sadrži rezultate veće od nule na temelju *VNClassificationObservation* koja, kao komponenta *Vision* biblioteke, sadrži informacije vezane za zahtjev procesiranja slike.

Funkcija *.map* vraća polje koje sadrži rezultate mapiranja na temelju opažanja koji se potom bilježe u decimalnom formatu uz pomoć *confidence* varijable koja daje podatak o pouzdanosti percipirane informacije.

Pohrana najnovije procjene, odnosno zapažanja vrši se putem *imeObjekta* varijable koja se na posljetku izjednačava sa *MLProcjena* varijablom čiji je zadatak, kako je na početku definirano, putem *String* strukture ispisati najnoviju *Core ML* procjenu percipiranog objekta.

Posljednja funkcija imena *updateCoreML()* sadrži sljedeću sintaksu:

```
func updateCoreML() {

      let privremeniSpremnik : CVPixelBuffer? =
(sceneView.session.currentFrame?.capturedImage)
      if privremeniSpremnik == nil { return }

      let ciImage = CIImage(cvPixelBuffer: privremeniSpremnik!)

      let upravljacZahtjevaSlike = VNImageRequestHandler(ciImage:
ciImage, options: [:])

      do {

            try upravljacZahtjevaSlike.perform(self.vizijaZahtjev)

            } catch {

            print(error)

            }
      }
}
```

CVPixelBuffer predstavlja referencu koja se odnosi na *Core Video pixel buffer* objekt koji pohranjuje sliku u glavnu memoriju. Tim slijedom, konstanta *privremeniSpremnik* sadrži vrijednosti trenutno percipirane scene putem *ARKit* biblioteke.

Konstanta *ciImage* uključuje *CIImage* koja predstavlja reprezentaciju slike koja se treba procesirati putem *Core Image* filtera, a sadrži *cvPixelBuffer* referencu koja izvlači sliku kao objekt iz sadržaja *Core Video pixel buffer* spremnika.

Za pripremu *Core ML* zahtjeva koristi se *upravljacZahtjevaSlike* konstanta koja pokreće *VNImageRequestHandler* čija se uloga odnosi na obradu jednog ili više zahtjeva za analizu slike.

Pokretanje zahtjeva za sliku vrši se uz pomoć *do – catch* izjave. S obzirom da zahtjev može izbaciti grešku, započinje se ključnom riječju *try*, nakon čega slijedi izvršenje na razini *Vision* zahtjeva pomoću *perform* funkcije. U slučaju greške, ista se ispisuje putem *print* funkcije.

Razlog primjene ove sintakse jest činjenica da se slika percipirana pomoću *ARKit* biblioteke bilježi u *YUV*[2] formatu, odnosno istu je potrebno konvertirati u *RGB*[3] vrijednosti kako bi se osigurala adekvatna reprezentacija na zaslonu uređaja, kao i kompatibilnost sa većim brojem *Core ML* modela računalnog učenja koji bi se potencijalno mogli upotrijebiti kasnije.

Na samom kraju upotrijebljena je sljedeća sintaksa:

```
extension UIFont {

      func saOsobinama (osobine:UIFontDescriptorSymbolicTraits...) ->
UIFont {

      let descriptor =
self.fontDescriptor.withSymbolicTraits(UIFontDescriptorSymbolicTraits
(osobine))

      return UIFont(descriptor: descriptor!, size: 0)
    }
}
```

Svrha navedenog koda jest primjena ranije korištene *traitBold* varijable za svrhe prikazivanja teksta sa manje primjetnim nedostatkom poligonalnih elemenata. Primjena navedenog koda se također odnosi i na ostale vrste tekstualne modifikacije kao što je primjerice opcija *italic*.

Tim slijedom, korištenjem *UIFontDescriptorSymbolicTraits* iz *UIKit* biblioteke implementirana je osobina interpretacije stilističkih aspekata određenog fonta u kombinaciji sa *UIFont* klasom koja sadrži *fontDescriptor* varijablu zaslužnu za vraćanje opisa fonta, te *withSymbolicTraits* funkciju koja vraća opis novog fonta sa detaljnim, odnosno specifičnim svojstvima.

Na posljetku, *UIFont* klasa vraća font na temelju opisa, tj. *descriptor* konstante.

[2] Sustav boja za kodiranje analognog signala pri čemu *Y* predstavlja osvjetljenje, dok se *U* i *V* odnose na boju (SoftPixel, n.d.)
[3] Sustav boja koji miješanjem crvene (*Red*), zelene (Green) i plave (*Blue*) boje prikazuje sliku na zaslonu uređaja (Rouse, 2005.)

7.3.3. Izrada vlastitog *Core ML* modela računalnog učenja

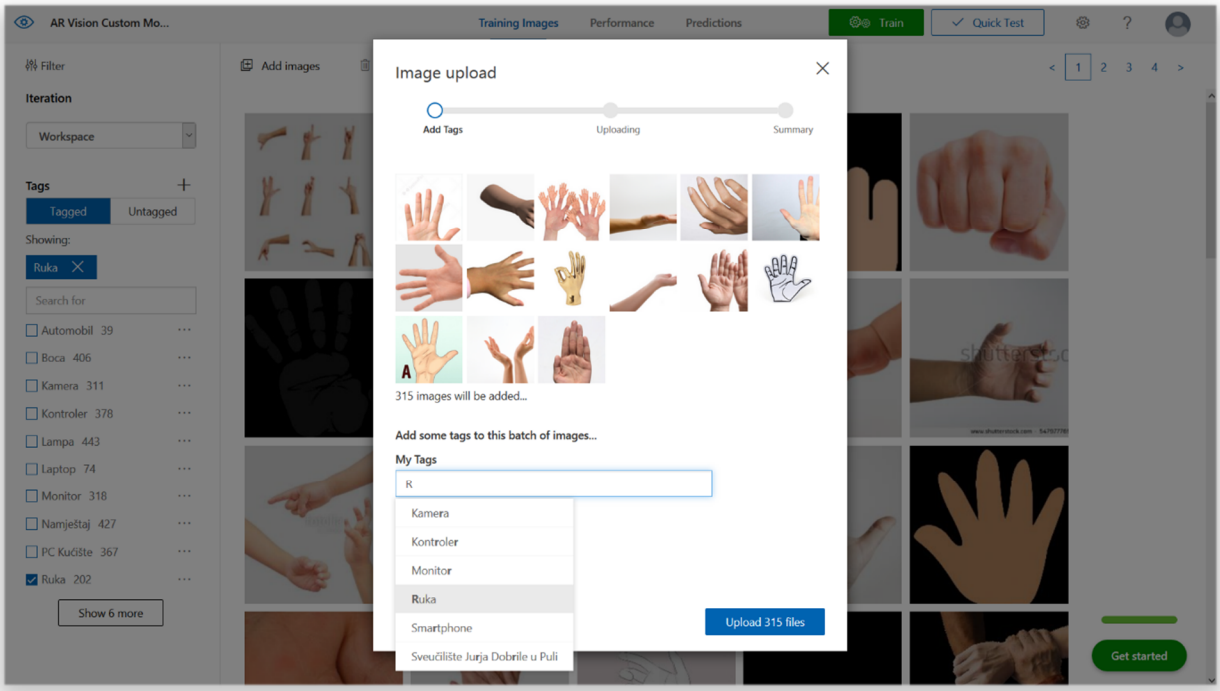
Za potrebe generiranja *Core ML* modela korišten je *Microsoft Cognitive* servis pod nazivom *Custom Vision* čija svrha jest pojednostavljenje procesa stvaranja modela sa manjom količinom podataka. Uz uvjet kreiranja *Microsoft* profila, *Custom Vision* nudi limitiranu besplatnu inačicu servisa koja uključuje kreiranje do dva vlastita projekta.

Kako bi se osigurala nepristranost rezultata, za svaku vrstu predmeta pohranjena je veća količina slika koje obuhvaćaju različite kuteve gledišta i upada svjetlosti, a kao izvor slika korištena je pretraga putem *Google* tražilice u kombinaciji sa dodatkom za serijsko preuzimanje slika u sklopu Web preglednika.

Radi preciznije percepcije pojedinih vrsta objekata, pohranjena je količina u rasponu od 100 do 500 slika. Svaka pojedina serija sadržavala je do 60 slika kako bi se postiglo adekvatnije treniranje algoritma za prepoznavanje. Pritom, valja napomenuti kako se za jednostavnije predmete koristi manja količina slika, dok je za kompleksnije poželjan što veći broj kako bi se na posljetku mogli preciznije raspoznati putem aplikacije.

Sve pohranjene slike s pratećim etiketama, za ispis naziva klase kojoj pripadaju, spadaju pod *General (compact)* domenu unutar *Microsoft Custom Vision* sučelja budući da navedena opcija jedina pruža mogućnost preuzimanja u formatu prikladnom za *iOS Core ML* modele.

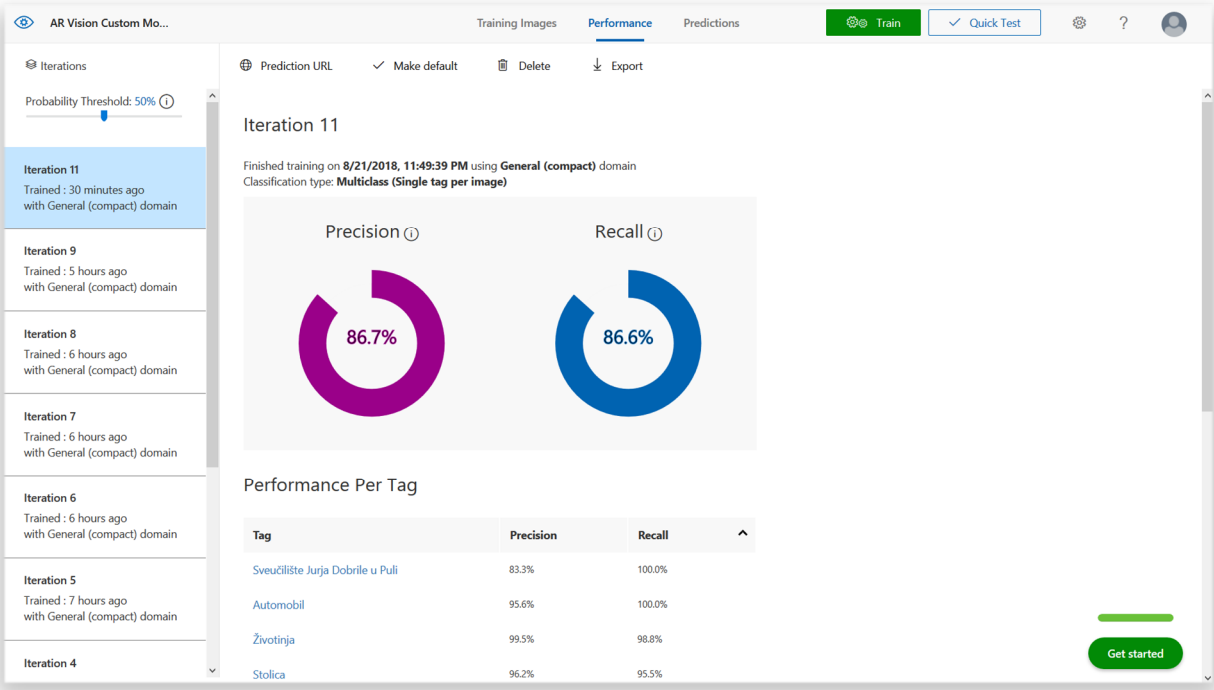
Za potrebe aplikacije pohranjeno je 1,05 GB slika uzimajući u obzir limit veličine u iznosu od 6 MB po zapisu. Slijedom toga, ispunjena je maksimalna kvota za besplatnu inačicu servisa koja obuhvaća 5000/5000 pohranjenih slika, 17/50 dodijeljenih etiketa, te 10/10 dozvoljenih iteracija za treniranje modela koje se temelji na percipiranju razlika među pohranjenim slikama.



Slika 43: Proces pohrane slika uz odgovarajuću etiketu

(Izvor: Autor)

Kako bi se uspostavile razlike među pohranjenim slikama, odnosno kategorijama predmeta korištena je *Train* opcija koja je nakon nekoliko trenutaka ispisala *Precision* (hrv. Preciznost) i *Recall* (hrv. Prisjećanje) vrijednosti koje prikazuju vjerojatnost točnosti rezultata na temelju modela i broj rezultata koje je model točno prepoznao izraženih u postocima. Pritom valja istaknuti da se postoci iznad 60% smatraju dostatnim, dok važniju ulogu igra raznovrsnost pohranjenih slika koja dovodi do preciznijih rezultata u konačnici.

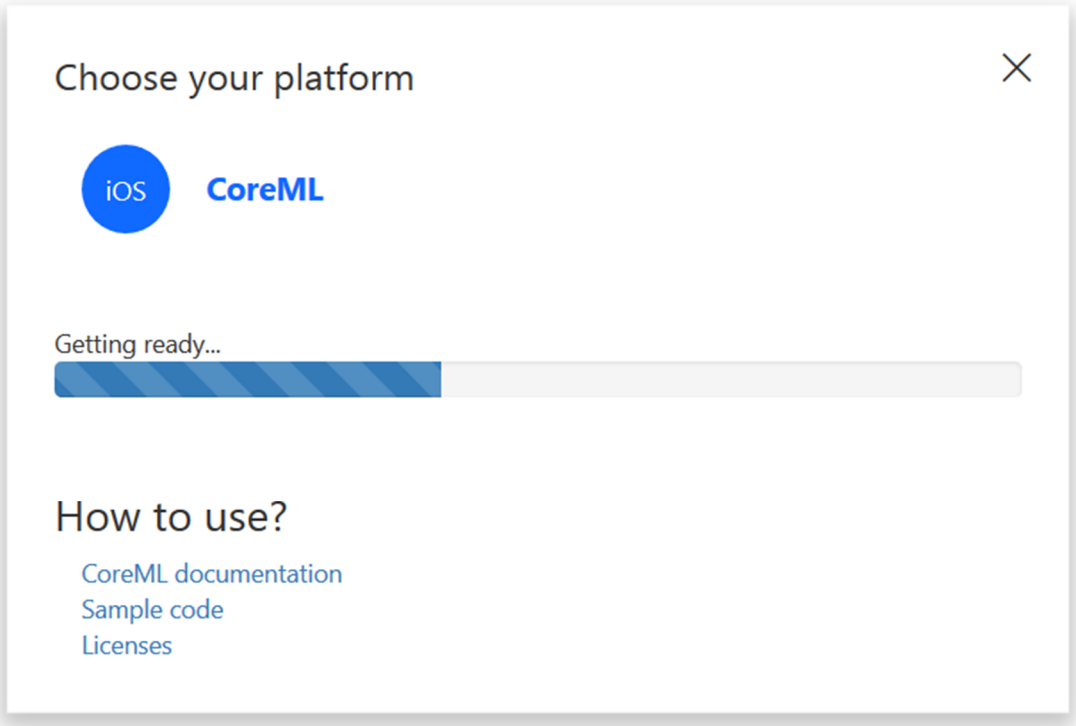


Slika 44: Prikaz učinkovitosti kreiranog modela računalnog učenja

(Izvor: Autor)

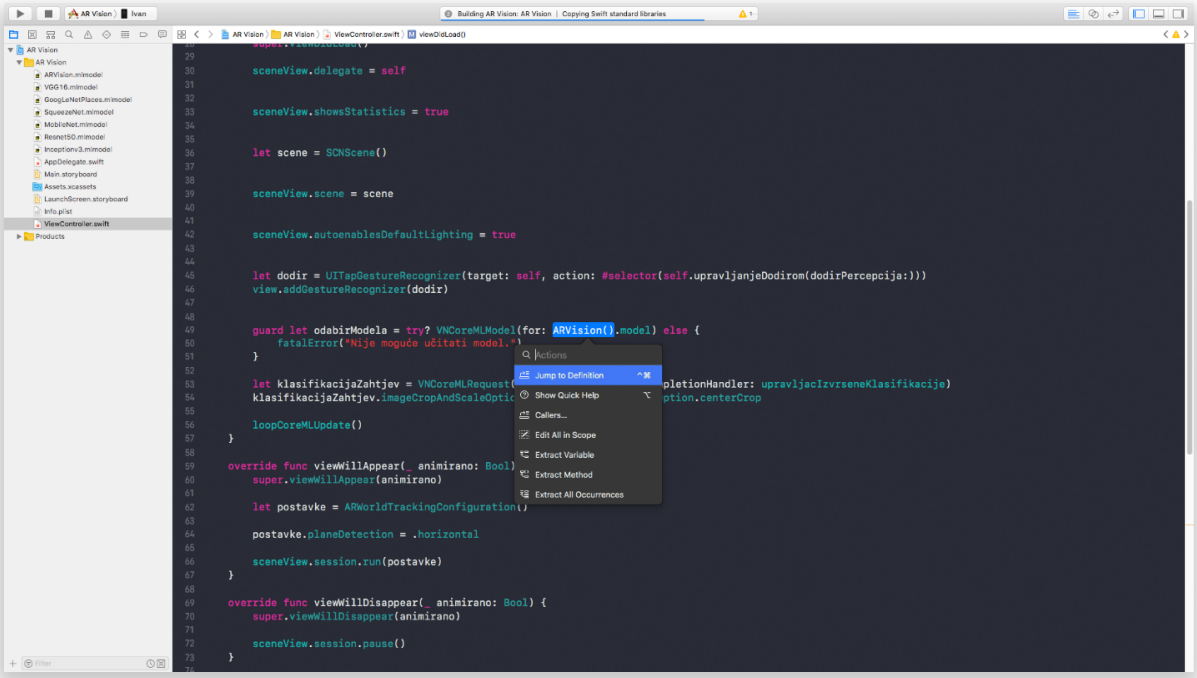
Nakon treniranja modela i kontroliranja performansi s ciljem potenciranja percepcije predmeta u raznolikom, tj. promjenjivom okruženju, kreirani model je eksportiran putem *Export* kratice. Pored popularnijih okvira kao što su *Tensorflow* za *Android* platformu, *ONNX* za *Windows*, te *DockerFile* za *IoT* (*Internet of Things*) uređaje, *Microsoft Custom Vision* također uključuje i podršku za *Core ML* modele čija upotreba je zastupljena na *Apple iOS 11* uređajima (Microsoft Corp., n.d.). Odabirom željene verzije pokreće se proces konverzije nakon kojeg se pojavljuje poveznica za preuzimanje kreiranog modela.

Kako bi se model mogao koristiti u kreiranoj aplikaciji, preimenovan je u *ARVision.mlmodel* i pohranjen unutar direktorija u kojem se nalaze ostale datoteke na temelju kojih se pokreće sama aplikacija. Također, nastalu je promjenu potrebno ažurirati u samom kodu aplikacije kako bi se na posljetku uspješno pokrenula.



Slika 45: Proces konverzije i eksportiranja kreiranog *Core ML* modela

(Izvor: Autor)



Slika 46: Proces implementiranja vlastitog *ARVision* modela unutar sintakse i pokretanje aplikacije na *iOS* uređaju putem *Xcode* sučelja

(Izvor: Autor)

7.3.4. Testiranje kreirane *Core ML* aplikacije

Prilikom pokretanja aplikacije može se zamijetiti kraći period bijelog zaslona tijekom kojeg se odvijaju *ARKit* kalkulacije koje uključuju detekciju horizontalnih i vertikalnih ploha, prepoznavanje dubine i udaljenosti uređaja od percipirane površine, te kompenziranje rasvjete na temelju okruženja.

Po završetku učitavanja esencijalnih elemenata za funkcionalnost, aplikacija je spremna za korisnički input, odnosno postavljanje točaka povrh predmeta u prostoru na temelju kojih će se vršiti kalkulacije pripadnosti određenim kategorijama. Imajući u vidu veličinu generiranih etiketa, odnosno slova, preporučljivo je korištenje na udaljenosti od 20 cm do 200 cm kako bi se mogao raspoznati generirani tekst.

Preciznije kalkulacije, odnosno pouzdaniji rezultati prepoznavanja predmeta javljaju se ukoliko se radi o manjoj udaljenosti između uređaja i željenog predmeta. Također, valja napomenuti kako generirane točke uzimaju parametre prostorne orijentacije uređaja stoga iste neće biti vidljive ukoliko se aplikacija primjerice koristi u vozilu koje je u pokretu. S obzirom na ograničenja aplikacije, poželjno je označavanje objekata koji su statični, odnosno čija se lokacija u prostoru ne mijenja značajno u odnosu na poziciju uređaja putem kojeg se percipiraju.

Pri dnu zaslona evidentna je prisutnost trake koja ukazuje na neke od relevantnih parametara glede funkcionalnosti same aplikacije. Za prikaz trake zaslužna je sljedeća linija koda na samom početku sintakse:

```
    sceneView.showsStatistics = true
```

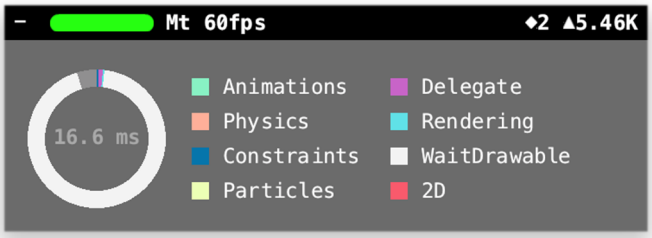
Zahvaljujući tome, dostupne su informacije vezane za grafičke performanse aplikacije na temelju parametara dostupnih putem grafičke procesorske jedinice uređaja.

Prvi vidljiv parametar odnosi se na broj slika po sekundi, odnosno brzinu osvježavanja slike. Iako se kao standardna brzina uzima 60 slika po sekundi, moguće je zamijetiti kako se ista smanjuje u skladu sa količinom trodimenzionalnog sadržaja generiranog na temelju korisničkog inputa.

Uzrok tome jest nekoliko ključnih parametara koji uključuju kompleksnost prikazanog fonta koja diktira broj poligonalnih elemenata za procesiranje, količina zasebnih trodimenzionalnih objekata čije generiranje diktira korisnik, te temperatura uređaja koja je u izravnoj korelaciji sa prethodno navedenim. Slijedom toga, pored uobičajenih *ARKit* i *SceneKit* kalkulacija koje se konstantno odvijaju, najčešća stopa osvježenja zaslona iznosi 30 slika po sekundi.

Sljedeći vidljiv parametar predstavlja broj percipiranih čvorova u sceni popraćen brojem poligonalnih elemenata. Broj čvorova se povećava u skladu sa brojem generiranih trodimenzionalnih objekata na temelju korisničkog inputa, što zauzvrat potencira rast broja poligonalnih elemenata u sceni koji ovisi od kompleksnosti pojedinih trodimenzionalnih objekata.

Kružni grafikon predstavlja udio pojedinih operacija u ukupnom procesiranju jedne slike na zaslonu, dok sredina kružnice predstavlja vremenski okvir, iskazan u milisekundama, unutar kojeg se spomenute operacije odvijaju.



Slika 47: Traka sa statistikama na temelju *SceneKit* biblioteke
(Izvor: Autor)

Pored navedenog, jedini preostali grafički element aplikacije jest centralna točka, kreirana na samom početku u sklopu *Main.Storyboard* datoteke putem *Xcode* sučelja, čija pojava u vidu + simbola korisniku predstavlja točku pri kojoj će se pojaviti, odnosno generirati sadržaj prilikom dodira zaslona.

Slika 48: Prikaz funkcije prepoznavanja predmeta pomoću kreirane aplikacije
(Izvor: Autor)

7.4. Aplikacija za prikaz *Google Places* interesnih točaka

U sklopu ovog poglavlja obrađena je izrada aplikacije koja na temelju pozicije korisnika identificira točke interesa na temelju trenutne lokacije i prikazuje ih povrh stvarnog okruženja uz pomoć integrirane kamere na uređaju.

Za definiranje točaka interesa (engl. *POI – Points Of Interest*) koristi se *Google Places API* sučelje u kombinaciji sa *HDAugmentedReality* bibliotekom kako bi se, u vidokrugu kamere, točke s pratećim udaljenostima uspješno prikazale.

7.4.1. Početne postavke

Nalik ranijim primjerima, prije svega je definiran razvojni tim, odnosno pripadajući developerski certifikat kako bi se aplikacija mogla pohraniti i pokrenuti na samom uređaju.

Unutar *Main.Storyboard* datoteke postavljen je *MapView* element čije dimenzije prekrivaju kompletan zaslon, te *UIButton* element imena *Kamera* pomoću kojeg će se kasnije pokretati kamera uređaja po potrebi.

HDAugmentedReality biblioteka je također uključena u strukturu glavnog direktorija, a pored toga kreirani su zapisi *MjestaUcitavanje.swift* i *Mjesto.swift* čija je uloga vraćanje rezultata upita u vidu interesnih točaka na temelju *Google Places API* sučelja, te bilježenje rezultata u pripadajuću klasu.

Prva važna stavka koju je potrebno definirati jest lokacija korisnika, stoga se koristi *CLLocationManager* klasa čija uloga podrazumijeva započinjanje i prekidanje dostave lokacijski orijentiranih događaja aplikaciji. Tim slijedom, unutar *ViewController* datoteke, definirana je *locationManager* konstanta.

Potom, dodana je ekstenzija *ViewController* datoteci koja uključuje *CLLocationManagerDelegate{}* protokol. Navedeni protokol sadrži metode koje se koriste za primanje poziva događaja pripadajućih objekata koji upravljaju lokacijskim parametrima.

Kako bi se odobrilo korištenje *GPS* odašiljača za prikupljanje lokacijskih parametara, unutar *Info.plist* zapisa dodan je ključ *NSLocationWHenInUseUsageDescription* pomoću kojeg je omogućeno iniciranje upita putem kojeg korisnik odobrava ili odbija korištenje spomenutog senzora. Kreirani upit se pojavljuje samo jednom prilikom inicijalnog pokretanja aplikacije, nakon čega ostaje trajno zapisan u postavkama uređaja.

Zahvaljujući kreiranim postavkama, moguće je dobiti podatak o lokaciji. Kako bi se to realiziralo, unutar *viewDidLoad()* korišten je sljedeći kod:

```
            super.viewDidLoad()
            locationManager.delegate = self
            locationManager.desiredAccuracy =
kCLLocationAccuracyNearestTenMeters
            locationManager.startUpdatingLocation()
            locationManager.requestWhenInUseAuthorization()
```

Uloga *delegate* varijable jest obavještavanje o promjeni pozicije uređaja, dok preciznost pozicije određuje *kCLLocationAccuracyNearestTenMeters* konstanta iz *CoreLocation* biblioteke čija je preciznost dostatna za potrebe trenutne aplikacije. Na posljetku *requestWhenInUseAuthorization()* funkcija korisniku postavlja upit za odobrenje pristupa lokacijskim parametrima ukoliko isto već nije omogućeno.

Za dohvaćanje trenutne lokacije koristi se sljedeći kod:

```
func locationManager(_ manager: CLLocationManager, didUpdateLocations
lokacije: [CLLocation]) {

  if lokacije.count > 0 {
    let lokacija = lokacije.last!

    if lokacija.horizontalAccuracy < 100 {
      manager.stopUpdatingLocation()
      let raspon = MKCoordinateSpan(latitudeDelta: 0.014,
longitudeDelta: 0.014)

      let regija = MKCoordinateRegion(center: lokacija.coordinate,
span: raspon)

      mapView.region = regija
    }
  }
```

Tijekom svakog osvježenja lokacije uz pomoć *locationManager* konstante, šalju se ažurirani parametri putem *delegate* varijable. Polje sa lokacijama sadrži sve lokacije u kronološkom redoslijedu, odnosno najnovija lokacija je zadnji objekt unutar polja. Prvo se vrši provjera ukoliko postoje lokacije u polju, a potom ukoliko se nađe barem jedna, uzima se najnoviji podatak.

Varijabla *horizontalAccuracy* bilježi i pohranjuje horizontalnu preciznost. Ova vrijednost predstavlja radijus oko trenutne lokacije iskazan u metrima, odnosno radijus od 100 metara oko pozicije pohranjene unutar *lokacija* konstante.

Potom, *if* izjava provjerava ukoliko je točnost dovoljno visoka za trenutne potrebe. Budući da je definirana vrijednost od 100 metara, postojana je mogućnost da proces prikupljanja parametara dosegne vremenski okvir duži od jedne minute.

Funkcija *stopUpdatingLocation()* obustavlja generiranje lokacijskih parametara, a njena glavna svrha jest štednja baterije uređaja budući da *GPS* odašiljač konzumira značajne količine resursa.

Konstante *raspon* i *regija* zaslužne su za zumiranje mape na trenutnu lokaciju korisnika, odnosno uređaja.

7.4.2. Implementiranje *Google Places API* sučelja

Zahvaljujući prethodno navedenom kodu, moguće je dohvaćanje trenutne lokacije, to jest omogućeno je dohvaćanje popisa interesnih točaka putem *Google Places API* sučelja.

Korištenje *Google Places API* sučelja iziskuje registraciju koja podrazumijeva polja kao što su ime, prezime, adresa, datum rođenja, pozivni broj, e-mail adresa, te broj aktivne kreditne kartice. Ukoliko se premaši limitirani broj upita putem sučelja, za daljnje korištenje servisa koristi se automatska naplata prema cjeniku dostupnom na službenoj stranici za developere.

Nakon kreiranja profila kreiran je novi projekt, a potom je aktivirana *Google Places API Web Service* opcija putem za to namijenjene poveznice. Kreiranjem spomenutog servisa, pod *Credentials* poveznicom preuzet je *API* ključ koji je potreban za prikazivanje *Google* servisa u sklopu same aplikacije.

API ključ implementiran je unutar *MjestaUcitavanje.swift* datoteke u sljedećem dijelu koda:

```
let apiURL = "https://maps.googleapis.com/maps/api/place/"
let apiKey = "AIzaSyBiLf1wcEZJpl4MFfIFCzL5pL6d5d_EVIs"
```

Nakon implementiranja ključa, kako bi se aplikacija pokrenula, unutar *ViewController.swift* datoteke dodane su sljedeće linije koda:

```
    fileprivate var mjesta = [Mjesto]()
    fileprivate var startedLoadingPOIs = false
```

Uz pomoć *startedLoadingPOIs* varijable omogućeno je praćenje zahtijeva u tijeku. Budući da se *CLLocationManagerDelegate* metoda može pozvati više puta nakon osvježenja lokacije, veći broj upita limitiran je pomoću varijable *mjesta* koja pohranjuje zaprimljene točke interesa (*POI*).

Nadalje, unutar *locationManager* funkcije dodana je sljedeća *if* izjava:

```
    if !startedLoadingPOIs {
      startedLoadingPOIs = true

      let ucitavanje = MjestaUcitavanje()
      ucitavanje.loadPOIS(location: lokacija, radius: 5000) { placesDict,
    error in

        if let dict = placesDict {}
      }
    }
```

Ovim putem pokrenuto je učitavanje popisa interesnih točaka koje se nalaze u radijusu od 5000 metara od trenutne lokacije uređaja prilikom čega se iste ispisuju u konzoli. Na taj način je omogućeno otklanjanje neispravnosti, odnosno u slučaju *null* ispisa radijus se postupno povećava kako bi se postiglo prikazivanje točaka u konačnici.

Slijedom navedenog, aplikacija je u stanju odrediti poziciju korisnika, učitati popis točaka interesa unutar zadanog radijusa i pohraniti mjesto unutar pripadajuće klase.

7.4.3. Prikazivanje točaka interesa (*POI*)

Kako bi se *mapView* varijabli dodijelila anotacija, kreirana je dodatna klasa unutar novo kreiranog zapisa pod imenom *MjestoAnotacija.swift* čija sintaksa sadrži sljedeće:

```
import Foundation
import MapKit

class MjestoAnotacija: NSObject, MKAnnotation {
  let coordinate: CLLocationCoordinate2D
  let naslov: String?

  init(lokacija: CLLocationCoordinate2D, naslov: String) {
    self.coordinate = lokacija
    self.naslov = naslov

    super.init()
  }
}
```

Na ovaj je način klasa implementirala *MKAnnotation* protokol zaslužan za povezivanje sadržaja sa specifičnom lokacijom na mapi, kao i dva svojstva popraćena *init* metodom. Sukladno tome, kreirani su svi potrebni elementi za prikazivanje interesnih točaka na mapi.

Nakon toga, unutar *ViewController.swift* zapisa dopunjena je *locationManager* metoda na sljedeći način:

```
guard let mjestaPolje = dict.object(forKey: "results") as?
[NSDictionary]  else { return }

              for mjestoPopis in mjestaPolje {
                let latitude = mjestoPopis.value(forKeyPath:
"geometry.location.lat") as! CLLocationDegrees
                let longitude = mjestoPopis.value(forKeyPath:
"geometry.location.lng") as! CLLocationDegrees
                let referenca = mjestoPopis.object(forKey: "reference")
as! String
                let ime = mjestoPopis.object(forKey: "name") as! String
                let adresa = mjestoPopis.object(forKey: "vicinity") as!
String

                let lokacija = CLLocation(latitude: latitude, longitude:
longitude)
                let mjesto = Mjesto(location: lokacija, reference:
referenca, name: ime, address: adresa)

                self.mjesta.append(mjesto)
                let anotacija = MjestoAnotacija(lokacija:
mjesto.lokacija!.coordinate, naslov: mjesto.mjestoIme)
                DispatchQueue.main.async {
                  self.mapView.addAnnotation(anotacija)
                }
              }
```

Pri samom početku, *guard* izjava provjerava da li odgovor sadrži očekivani format. Potom, konstanta *mjestoPopis* iterira kroz dohvaćene točke interesa prilikom čega konstante *latitude*, *longitude*, *referenca*, *ime*, *adresa* i *lokacija* izdvajaju relevantne podatke iz primljenog odgovora nakon čega se kreira *Mjesto* objekt sa točnim podacima i pohranjuje unutar polja *mjesta*.

Konstanta *anotacija* zaslužna je za kreiranje *MjestoAnotacija* klase koja rezultira u prikazu anotacija na mapi unutar aplikacije. Kako bi se prikaz na mapi realizirao, putem *DispatchQueue.main.async* naredbe anotacija je dodana *mapView* varijabli.

Konačni rezultat navedenog jest pojavljivanje anotacija na grafičkom prikazu mape na uređaju koji prima input, odnosno prilikom dodira anotacije ispisuje dostupne podatke kao što su naziv, Web stranica, telefonski broj i slično.

7.4.4. *HDAugmentedReality* biblioteka

Kako bi se implementirao element proširene stvarnosti u sklopu aplikacije, *UIButton* elementu imena *Kamera*, definiranom pri samom početku, potrebno je pridružiti određene operacije da bi konačan rezultat mogao prikazati okruženje korisnika omeđeno točkama interesa dohvaćenim sa *Google* servera.

Kako bi se navedeni proces pojednostavio, korištena je *HDAugmentedReality* biblioteka koja je zaslužna za prikaz slike putem kamere, kao i prekrivanje vidnog polja relevantnim sadržajem, odnosno točkama interesa u ovom slučaju. Također, biblioteka je zaslužna i za kalkuliranje udaljenosti točaka interesa u odnosu na trenutnu lokaciju uređaja što uvelike pojednostavljuje proces prikazivanja i izrade same aplikacije u konačnici.

Za kalkuliranje udaljenosti točaka, odnosno x i y koordinata pojedine točke prema Kartezijevom sustavu, uzimaju se geografska dužina i širina. Geografska dužina daje podatak o orijentaciji u smjeru istoka ili zapada u odnosnu na Greenwich kao centralnu točku, a iskazuje se u rasponu od –180 do +180 stupnjeva. Geografska širina govori da li se točka nalazi sjeverno ili južno u odnosu na ekvator, a raspon seže od +90 stupnjeva na Sjevernom polu do –90 stupnjeva na Južnom polu (NAAP Labs, n.d.).

Kako bi se preciznije izračunala udaljenost točaka, koristi se formula ortodromske udaljenosti koja ukazuje na najkraći put između dviju točaka na površini kugle. Navedena formula daje precizne rezultate uz odstupanje od 0.5%, odnosno manja udaljenost točaka daje bolju preciznost rezultata (Tseng et al., 2013.).

Unutar standardne *CLLocation* biblioteke može se naći *distanceFromLocation:* metoda koja izračunava udaljenost na identičnom principu, a ista se koristi i unutar *HDAugmentedReality* biblioteke.

Razlog implementacije *HDAugmentedReality* biblioteke jest kalkuliranje pozicije prikaza točaka interesa na zaslonu uređaja. Kalkulacija se vrši na osnovu pozicije uređaja u prostoru, odnosno ukoliko je uređaj primjerice usmjeren prema podu, točke interesa će biti prikazane uz gornji rub zaslona ili neće biti prikazane uopće. Stoga, može se zaključiti kako na temelju spomenute biblioteke anotacije za točke interesa ostaju relativno fiksirane u prostoru na temelju njihove lokacije neovisno o usmjerenju kamere uređaja.

Navedeno ponašanje točaka rezultat je *HDAugmentedReality* kalkulacija koje se vrše na temelju informacija prikupljenih od strane žiroskopskog senzora i kompasa. Koristeći podatke kao što su smjer gledanja i stupanj nagiba uređaja moguće je izračunati na kojem dijelu zaslona je potrebno prikazati anotacije za najbliže točke interesa.

Slijedom navedenog, uloga i važnost *HDAugmentedReality* biblioteke su evidentni, a njena primjena kod izrade aplikacija baziranih na lokacijski orijentiranim servisima sa proširenom stvarnošću je gotovo neizbježna.

7.4.5. Implementiranje *HDAugmentedReality* biblioteke

Unutar *HDAugmentedReality* biblioteke nalaze se sljedeće *.swift* datoteke:

- *ARAnnotation* – služi za definiranje točke interesa
- *ARAnnotationView* – kreira pogled za točke interesa
- *ARConfiguration* – definira osnovne postavke i pomoćne metode
- *ARTrackingManager* – vrši kalkulacije udaljenosti
- *ARViewController* – upravlja prikazom video signala i dodaje anotacije u vidljiv spektar

Kako bi ispravno funkcionirao, *ARViewController* implementiran je u sklopu *ViewController* datoteke na sljedeći način:

```
var arViewController: ARViewController!
```

Potom, unutar *showARController* funkcije dodana je sljedeća sintaksa:

```
@IBAction func showARController(_ sender: Any) {
    arViewController = ARViewController()
    arViewController.dataSource = self
    arViewController.maxDistance = 0
    arViewController.maxVisibleAnnotations = 30
    arViewController.maxVerticalLevel = 5
    arViewController.headingSmoothingFactor = 0.05

    arViewController.trackingManager.userDistanceFilter = 25
    arViewController.trackingManager.reloadDistanceFilter = 75
    arViewController.setAnnotations(mjesta)
    arViewController.uiOptions.debugEnabled = false
    arViewController.uiOptions.closeButtonEnabled = true

    self.present(arViewController, animated: true, completion: nil)
  }
```

Postavljanjem *dataSource* varijable uz *arViewController* omogućeno je pregledavanje percipiranih točaka interesa.

Putem *maxVisibleAnnotations* varijable definiran je maksimalan broj koji podrazumijeva do 30 istovremeno vidljivih anotacija u odnosnu na maksimalno dozvoljenih 500. Razlog nižeg broja jest fluidnija funkcionalnost same aplikacije.

Varijabla *headinsSmoothingFactor* korištena je za kontroliranje kretanja interesnih točaka na zaslonu. Kako bi se osiguralo fluidno kretanje točaka definirana je vrijednost manja od 1. Manja vrijednost rezultira u animiranom kretanju uz popratni efekt zaostajanja anotacija u odnosu na kretanje uređaja. Slijedom navedenog, odabrana je optimalna vrijednost od 0.05 jedinica.

Pomoću *maxDistance* varijable definirana je udaljenost raspona prikaza pogleda u metrima, odnosno sve izvan zadanog parametra se neće prikazivati.

U konačnici, *arViewController* varijabla pozvana je za prikazivanje uz popratnu funkciju animacije.

7.4.6. Implementiranje *ARDataSource* metode

Nadalje, kako bi se uklonila nastala greška unutar *Xcode* sučelja, *ViewController* datoteci je dodan *ARDataSource* protokol koji *ViewController* datoteci prosljeđuje informacije potrebne za prikaz anotacija, odnosno točaka interesa. Slijedom toga, kreirana je nova datoteka pod imenom *AnnotationView.swift*.

AnnotationView.swift datoteka sadrži sljedeću sintaksu:

```
import UIKit

protocol AnnotationViewDelegate {
  func didTouch(annotationView: AnnotationView)
}


class AnnotationView: ARAnnotationView {
  var nazivLabel: UILabel?
  var udaljenostLabel: UILabel?
  var strelicaLabel: UILabel?
  var delegate: AnnotationViewDelegate?

  override func didMoveToSuperview() {
    super.didMoveToSuperview()

    loadUI()
  }


  func loadUI() {
    nazivLabel?.removeFromSuperview()
    udaljenostLabel?.removeFromSuperview()
    strelicaLabel?.removeFromSuperview()

    let strelica = UILabel(frame: CGRect(x: 5, y: 5, width: 5, height:
30))
    strelica.backgroundColor = UIColor (red: 0.126, green: 0.582, blue:
1.000, alpha: 1.000)
    strelica.transform = CGAffineTransform(rotationAngle: CGFloat.pi /
4)


    let etiketa = UILabel(frame: CGRect(x: 10, y: 0, width:
self.frame.size.width, height: 30))
    etiketa.font = UIFont.boldSystemFont(ofSize: 14)
    etiketa.numberOfLines = 0
    etiketa.backgroundColor = UIColor(red: 0.844, green: 0.886, blue:
0.899, alpha: 0.900)
    etiketa.textColor = UIColor.black
    if #available(iOS 11.0, *) {
```

```
      etiketa.layer.maskedCorners = [.layerMinXMinYCorner,
.layerMaxXMinYCorner]
    } else {
      // Vracanje na prijasnju verziju
    }

    self.addSubview(etiketa)
    self.nazivLabel = etiketa

    self.addSubview(strelica)
    self.strelicaLabel = strelica

    udaljenostLabel = UILabel(frame: CGRect(x: 10, y: 10, width:
self.frame.size.width, height: 30))
    udaljenostLabel?.backgroundColor = UIColor(red: 0.126, green: 0.582,
blue: 1.000, alpha: 1.000)
    udaljenostLabel?.textColor = UIColor.white
    udaljenostLabel?.font = UIFont.boldSystemFont(ofSize: 12)
    udaljenostLabel?.layer.cornerRadius = 5.0
    udaljenostLabel?.layer.masksToBounds = true
    udaljenostLabel?.textAlignment = .center
    if #available(iOS 11.0, *) {
      udaljenostLabel?.layer.maskedCorners = [.layerMaxXMaxYCorner,
.layerMinXMaxYCorner]
    } else {
      // Vracanje na prijasnju verziju
    }
    self.addSubview(udaljenostLabel!)

    if let annotation = annotation as? Mjesto {
      nazivLabel?.text = annotation.mjestoIme
      udaljenostLabel?.text = String(format: "%.2f km",
annotation.distanceFromUser / 1000)

    }
     }
```

Pri samom početku dodan je *delegate* protokol, nakon čega je kreirana podklasa *ARAnnotationView* koja se koristi za prikazivanje pogleda točaka interesa unutar kojeg se nalaze etikete sa imenom i udaljenošću.

Potom, pozvana je *loadUI()* funkcija koja dodaje i definira postavke etiketa. Među definiranim etiketama nalaze se *nazivLabel*, *udaljenostLabel* i *strelicaLabel* varijable čija je uloga vizualiziranje pozadine okvira za naziv i udaljenost interesnih točaka.

Konstanta *strelica* ubačena je kako bi se vizualno dočarala lokacija na koju ukazuje anotacija. Da bi se stvorio privid strelice, korištena je *CGRect* struktura za kreiranje kvadrata koji je potom rotiran za 45 stupnjeva i postavljen na odgovarajuću poziciju. Rotacija kvadrata postignuta je putem *rotationAngle* naredbe koja kut rotacije prikazuje u radijanima. Iz tog razloga, *pi* vrijednost koja predstavlja 180 stupnjeva podijeljena je sa četiri kako bi se kao rezultat prikazala

rotacija od 45 stupnjeva. Zahvaljujući rotaciji, polovica kreiranog kvadrata utjelovljuje ulogu strelice pri dnu anotacije.

Konstanta *etiketa* također sadrži *CGRect* element, odnosno pravokutnik. Pored uobičajenih postavki veličine fonta i boje pozadine, pomoću *if* izjave primijenjeno je zaobljenje kuteva pravokutnika. Budući da se za zaobljenje koristi *Core Animation* biblioteka koja je podržana na *iOS 11* i novijim iteracijama operativnog sustava, putem *if* izjave kreiran je upit koji izvršava naredbu *maskedCorners* sa zadanim parametrima ukoliko verzija operativnog sustava odgovara zadanoj, odnosno *iOS 11* ili novijoj. Pomoću *layerMinXminYCorner* i *layerMaxXMinYCorner* varijabli iz *Core Animation* biblioteke postavljeno je zaobljenje dvaju gornjih kuteva etikete, tj. pravokutnika.

U slučaju *udaljenostLabel* varijable koja predstavlja donju polovicu, odnosno odvojeni pravokutnik za prikaz udaljenosti, također je upotrijebljena *CGRect* struktura za kreiranje pravokutnika uz dodatnu *if* izjavu za zaobljenje kuteva nalik prethodnoj. Pomoću *layerMaxXMaxYCorner* i *layerMinXMaxYCorner* varijabli zaobljeni su donji lijevi i donji desni kutevi prema prethodno definiranom radijusu od 5.0 jedinica.

Kako bi se dovršila *LoadUI()* funkcija, na posljetku je dodana sljedeća sintaksa:

```
override func layoutSubviews() {
    super.layoutSubviews()
    nazivLabel?.frame = CGRect(x: 10, y: 0, width:
self.frame.size.width, height: 40)
    nazivLabel?.layer.cornerRadius = 5.0
    nazivLabel?.layer.masksToBounds = true
    nazivLabel?.textAlignment = .center
    udaljenostLabel?.frame = CGRect(x: 10, y: 40, width:
self.frame.size.width, height: 20)
    strelicaLabel?.frame = CGRect(x: 65, y: 49, width: 28, height: 10)
  }

  override func touchesEnded(_ touches: Set<UITouch>, with event:
UIEvent?) {
    delegate?.didTouch(annotationView: self)
  }
}
```

Funkcija *layoutSubviews()* poziva se prilikom svake ponovne potrebe za prikazom pogleda, a sadrži definirane parametre za okvire koji obuhvaćaju *nazivLabel*, *udaljenostLabel* i *strelicaLabel* varijable, odnosno etikete.

U konačnici, pomoću *touchesEnded* funkcije iz *UIKit* biblioteke, *delegate* prima informaciju o dodiru zaslona na temelju čega se vrši procjena o tome da li se treba izvršiti određena operacija, te ukoliko je potrebno, odabire relevantna operacija za izvršenje.

Nakon toga, unutar *ViewController* datoteke dodana je sljedeća ekstenzija:

```
extension ViewController: AnnotationViewDelegate {
  func didTouch(annotationView: AnnotationView) {
    if let annotation = annotationView.annotation as? Mjesto {
      let placesLoader = MjestaUcitavanje()
      placesLoader.loadDetailInformation(forPlace: annotation) {
resultDict, error in

        if let infoDict = resultDict?.object(forKey: "result") as?
NSDictionary {
          annotation.telBr = infoDict.object(forKey:
"formatted_phone_number") as? String
          annotation.website = infoDict.object(forKey: "website") as?
String

          self.showInfoView(zaMjesto: annotation)
        }
      }

    }
  }
}
```

Na ovaj način *ViewController* nasljeđuje *AnnotationViewDelegate* protokol koji prikazuje *annotationView* anotaciju kao *Mjesto* klasu nakon čega se putem *MjestaUcitavanje* konstante učitavaju dodatne informacije za zadano mjesto. Na posljetku, dodijeljena su odgovarajuća svojstva putem *NSDictionary* klase iz *Foundation* biblioteke.

Nakon kreiranja navedenog, pri početku *ViewController* zapisa unutar *UIViewController* klase dodana je sljedeća sintaksa:

```
func showInfoView(zaMjesto place: Mjesto) {
    let obavijest = UIAlertController(title: place.mjestoIme , message:
place.infoText, preferredStyle: UIAlertControllerStyle.alert)
    obavijest.addAction(UIAlertAction(title: "OK", style:
UIAlertActionStyle.default, handler: nil))

    arViewController.present(obavijest, animated: true, completion: nil)
  }
```

Kako bi se prikazale dodatne informacije nakon dodira anotacije, kreirana je *obavijest* konstanta koja sadrži *UIAlertController* klasu sa nazivom i informativnim tekstom pripadajuće točke interesa.

Budući da *ViewController* nije dio *showInfoView* hijerarhije, za potrebe prikaza obavijesti korištena je varijabla imena *arViewController*.

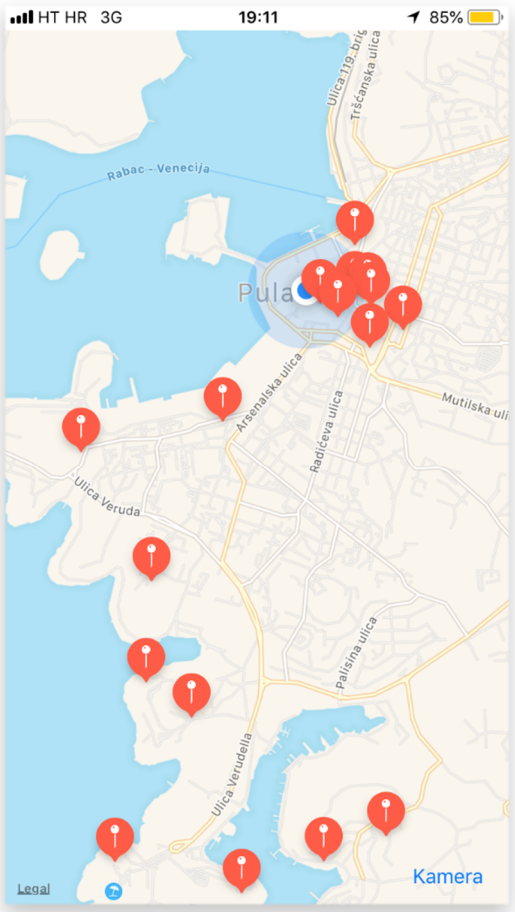
Budući da je ovime okončan kompletan kod aplikacije, prije pokretanja potrebno je unutar *Info.plist* datoteke dodati ključ *NSCameraUsageDescription* sa odgovarajućim opisom kako bi

se korisniku dalo do znanja da prilikom pokretanja *UIButton* poveznice imena *Kamera* odobrava korištenje stražnje kamere za potrebe prikaza proširene stvarnosti.

7.4.7. Testiranje kreirane aplikacije

Pri inicijalnom pokretanju aplikacije može se zamijetiti obavijest koja korisniku da je na znanje da zahtijeva korištenje *GPS* odašiljača za potrebe lociranja na prikazanoj mapi, odnosno zahtijeva dopuštenje od korisnika kako bi se spomenuti senzor mogao koristiti. Također, ukoliko pri kasnijem pokretanju aplikacije *GPS* odašiljač nije aktivan, korisniku se na zaslonu prikazuje obavijest koja poveznicom upućuje na odgovarajuću destinaciju unutar postavki uređaja kako bi se osiguralo aktiviranje samog senzora.

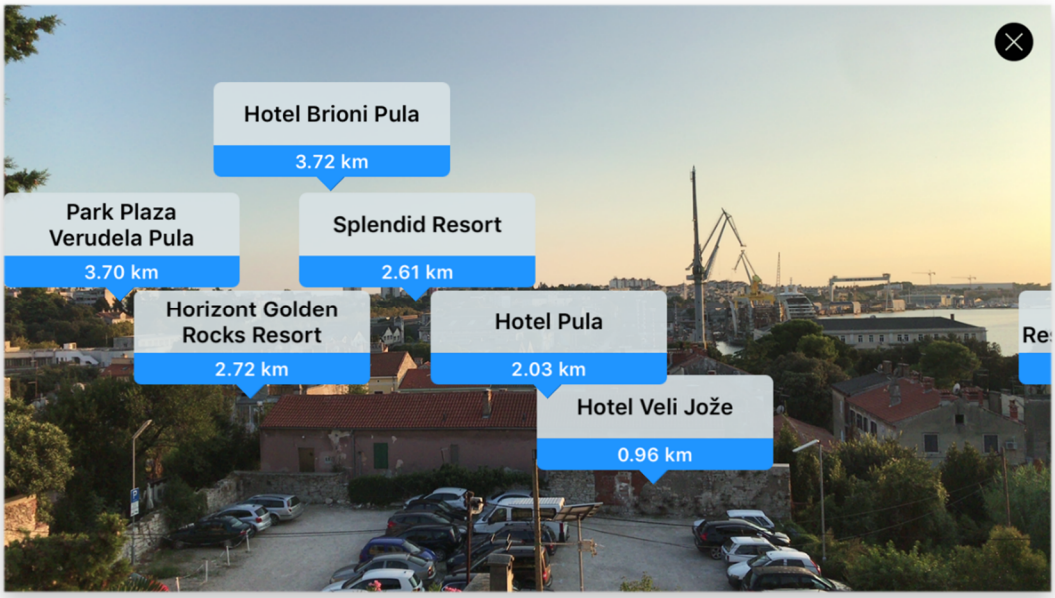
Aktiviranjem *GPS* senzora, lokacija korisnika se promptno prikazuje na mapi, a nedugo zatim se počinju pojavljivati točke interesa u skladu s prethodnom zadanim radijusom od 5000 metara. Valja istaknuti kako se povećavanjem mape na zaslonu proširuje prikazana površina što rezultira u pojavi dodatnih točaka interesa, a prednost prikazivanja točaka temelji se na njihovoj popularnosti percipiranoj od strane *Google Maps* servera. U skladu s time, korištenje aplikacije iziskuje povezivanje uređaja na Internet putem *Wi-Fi* signala ili mobilnog Interneta.



Slika 49. Početni zaslon s prikazom trenutne lokacije (plavo) i najbližih točaka interesa (crveno) (Izvor: Autor)

Dodirom poveznice pod imenom *Kamera*, pri donjem desnom kutu zaslona, korisniku se prilikom prvog pokretanja prikazuje obavijest za potvrdu korištenja stražnje kamere uređaja nakon čega se pojavljuje vidokrug kamere putem kojeg je aktivirana mogućnost percipiranja lokacije interesnih točaka u okolini.

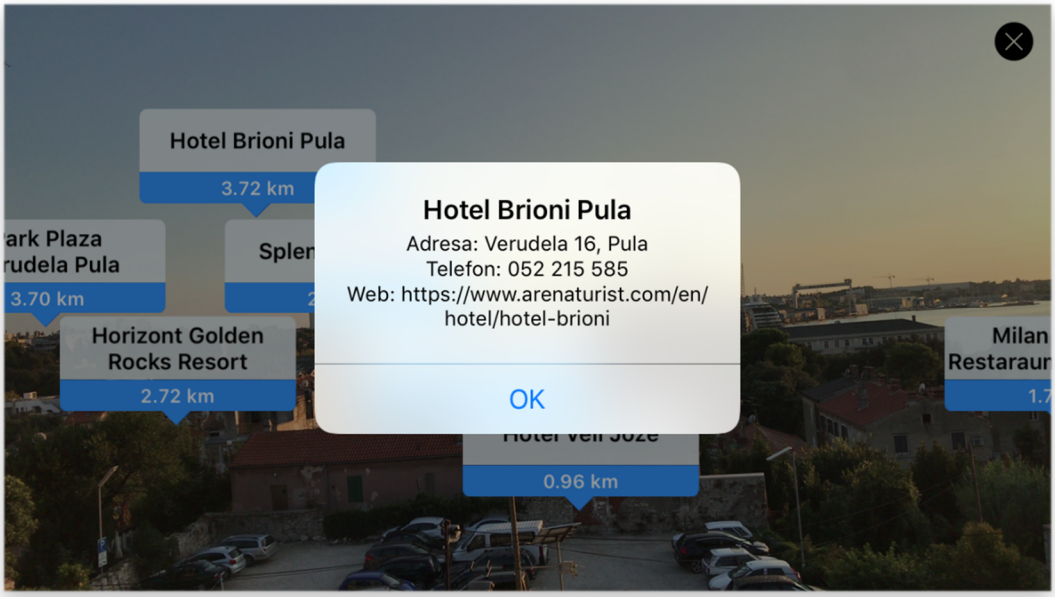
Dakako, prikaz točaka ovisi od pozicije uređaja u prostoru stoga je korisnik dužan usmjeriti uređaj u odgovarajućem pravcu kako bi se prikazale točke koje su locirane u tom smjeru na udaljenosti do 5000 metara. Izlazak iz prikaza pomoću kamere, odnosno proširene stvarnosti, moguć je putem za to namijenjenog gumba lociranog u gornjem desnom kutu zaslona.



Slika 50. Prikaz točaka interesa i njihove udaljenosti pomoću proširene stvarnosti (Izvor: Autor)

Kretanjem kroz prostor korisniku se mogu prikazati i dodatne točke koje inicijalno nisu bile vidljive. Razlog tome jest ranije navedena prednost definirana od strane *Google Maps* servera čiji tipovi kategorija prikazanih točaka interesa uključuju spektar od 90 različitih vrsta ustanova. Specifično prikazivanje određenih kategorija može se zadati unutar koda naredbom *types=*, a ukoliko ista nije definirana korisniku se prikazuju sve dostupne kategorije na trenutnoj lokaciji.

Dodirom anotacija za željenu točku interesa korisniku se prikazuje obavijest koja sadrži naziv ustanove, te polja adrese, telefona i Web stranice. Obavijest se pojavljuje povrh vidljivog spektra kamere dok anotacije i dalje ostaju vidljive u pozadini, a samu obavijest moguće je ukloniti iz prvog plana dodirom poveznice *OK*. Pritom valja istaknuti kako sve anotacije ne sadrže jednaku količinu podataka o odabranoj točki interesa, već je ista definirana na temelju podataka dostupnih na *Google Maps* serveru.



Slika 51. Prikaz informacija dostupnih pri dodiru željene anotacije

(Izvor: Autor)

ZAKLJUČAK

Od samog začeća ideje devedesetih godina 20. stoljeća do danas proširena stvarnost i njene primjene u raznim aspektima su, kroz niz iteracija, dostigle razinu pri kojoj je njena uloga nezaobilazna. Potpomognut suvremenim tehnološkim postignućima, prikaz virtualnog sadržaja u realnom okruženju dostigao je zavidnu razinu korisničkog interesa što nagovješćuje sve češću upotrebu proširene stvarnosti u sklopu nadolazećih uređaja i usluga. Budući da je sveprisutna primjena digitalnog sadržaja u današnjem svijetu neosporiva i mnoge djelatnosti ovise o istoj, a kolektivna svijest konzumenata spomenutog sadržaja kalibrirana za predstavljanje nove razine interaktivnog oblika upotrebe, primjena proširene stvarnosti više nije konceptualna materija već gotovo zagarantiran smjer razvoja dostupnih tehnologija.

Zahvaljujući širokom obimu mogućnosti koje proširena stvarnost pruža diljem raznovrsnih djelatnosti i popularizaciji podržanih servisa putem Interneta, baza korisnika je sve obimnija, a niz inovativnih implementacija sve brojniji. Suvremena rješenja, upotrebom svih raspoloživih resursa i ranije stečenih saznanja, sve promptnije nadograđuju pripadajuće platforme i time developerima pružaju opširniju lepezu dostupnih alata za razvoj, dok u mobilnoj sferi alati kao što su *Vuforia*, *Google AR Core*, te *Apple ARKit* pojednostavljuju način implementacije i, zahvaljujući dobro potkovanoj marketinškoj kampanji, usmjeravaju cjelokupno tržište mobilnih platformi ka nezaobilaznoj upotrebi inovativnih aplikacija koje će uz pomoć implementacije proširene stvarnosti korisnicima prijenosnih uređaja otvoriti nove vidike glede konzumacije i interakcije sa stvarnošću prožetom digitalnim sadržajem.

Neki od inovativnih koncepata primjene proširene stvarnosti materijalizirani su tijekom procesa sastavljanja ovog rada, a njihova demonstracija praktične primjene pritom služi kao testament funkcionalnosti i potencijala sinergije mobilne tehnologije, *iOS* platforme i proširene stvarnosti kao načina interakcije i konzumiranja sadržaja.

Na temelju predstavljenih koncepata koji ciljano kombiniraju sadržaj koji je od interesa korisnicima, odnosno potencijalno zanimljiv i relevantan, sa suvremenim tehnološkim inovacijama i nadogradnjama može se zaključiti kako proširena stvarnost predstavlja evidentan potencijal, odnosno upotrebljivu i korisnu alternativu za mnoštvo ranije uspostavljenih načina interakcije.

LITERATURA

[1] Antoniac P., *Augmented Reality Based user interface for mobile applications and services*, Faculty of Science, Department of Information, Processing Science, University of Oulu, Oulu, 2005.

[2] Azuma R. T., *A survey of augmented reality*, Presence: Teleoperators and Virtual Environments, vol. 6, no. 4, 1997.

[3] Azuma R., Baillot Y, Behringer R, Feiner S, Julier S, MacIntyre B: *Recent Advances in Augmented Reality*, IEEE Computer Graphics and Applications, Vol.21, No.6., 2001.

[4] Azuma R., *Location-Based Mixed and Augmented Reality Storytelling, Chapter 11 in 2 Edition of Fundamentals of Wearable Computers and Augmented Reality*, Woodrow Barfield (editor), CRC Press, 2015.

[5] Bajura M., Fuchs H., Ohbuchi R., *Merging virtual objects with the real world: Seeing ultrasound imagery within the patient*, in *Proc. 19th Annu. Conf. On Computer Graphics and Interactive Techniques*, ACM SIGGRAPH Computer Graphics, Volume 26, Issue 2, 1992.

[6] Bimber O., Raskar R., *Spatial Augmented Reality: Merging Real and Virtual Worlds*, A. K. Peters Ltd., Natick, 2005.

[7] Breen D. E., Rose E., Whitaker R. T., Tuceryan M., *Interactive Occlusion and Automatic Object Placement for Augmented Reality*, Proceedings of Eurographics, Futuroscope - Poitiers, France, 1996.

[8] Broll W., Lindt I., Ohlenburg J., Wittkämper M., Yuan C., Novotny T., Schieck A. V., Strothmann C. M. A. (2004), *A Collaborative Augmented Environment for Architectural Design and Urban Planning, Journal of Virtual Reality and Broadcasting*, Volume 1, no. 1, 2004.

[9] Bulearca, M., Tamarjan, D., *Augmented Reality: A Sustainable Marketing Tool?,* Global Business & Management Research, Vol. 2, 2010.

[10] Caudell T., Mizell D., *Augmented reality: An application of heads-up display technology to manual manufacturing technology augmented reality*, in Proc. Hawaii Int. Conf. on Syst. Science, 1992.

[11] Cawood S., Fiala M., *Augmented Reality: A Practical Guide*, Pragmatic Bookshelf, 2008.

[12] Edwards C., *Better than reality?*, Engineering & Technology, Volume 8, Issue 4, 2013.

[13] Henrysson A., Billingurst M., Ollila M., *AR Tennis*, Boston, 2006.

[14] Kato H., Billinghurst M., Poupyrev I., Imamoto K., Tachibana K., *Virtual Object Manipulation on a Table-Top AR Environment, Proceedings of the International Symposium on Augmented Reality*, Germany, 2000.

[15] Kaufmann H., *Collaborative augmented reality in education*, Institute of Software, Technology and Interative Systems, Vienna University of Technology, Wien, 2003.

[16] Kaufmann H., Schmalstieg D., *Mathematics and geometry education with collaborative augmented reality*, SIGGRAPH 2002 conference abstracts and applications, San Antonio, 2002.

[17] Kerawalla L., Luckin R., Seljeflot S., Wooland A., „Making it Real": Exploring the potential of Augmented Reality for teaching primary school science, University of Sussex, East Sussex, 2006.

[18] Krevelen D. W. F, Poelman R., *A Survey of Augmented Reality: Technologies, Applications, and Limitations*, The International Journal of Virtual Reality, Systems Engineering Section, Delft University of Technology, Delft, The Netherlands, 2010.

[19] Langlotz T., Grubert J., Grasset R., *Augmented reality Browsers: Essential products or only Gadgets?*, Communications of the ACM, Vol. 56, No 11, 2013.

[20] Leem K., *Augmented Reality in Education and Training*, Volume 56, Issue 2, University of Northern Colorado, 2012.

[21] Liarokapis F., Mourkoussis, N., White M., Darcy J., Sifniotis M., Petridis P., *Web 3D and augmented reality to support engineering education*, *World Transactions on Engineering and Technology Education*, University of Sussex, Falmer, 2004.

[22] Livingston M. A., Rosenblum L. J., Julier S. J., Brown D., Baillot Y., Swan J. E. II, Gabbard J. L., Hix D., *An Augmented Reality System for Military Operations in Urban Terrain*, Proceedings of Interservice / Industry Training, Simulation & Education Conference (I/ITSEC), December 2-5, Orlando, Florida, 2002.

[23] Mackay W., Velay G., Carter K., Ma C., Pagani D., *Augmenting reality: Adding computational dimensions to paper*, Commun. ACM, vol. 36, no. 7, 1993.

[24] Mann S., Wearable Computing: A First Step Toward Personal Imaging, IEEE Computer Society Press, Los Alamitos, 1997.

[25] Milgram P., Takemura H., Utsumi A., Kishino F., *Augmented reality: A class of displays on the reality-virtuality continuum*, ATR Communication Systems Research Laboratories, Kyoto, 1994.

[26] Milgram P., Kishino F., *A Taxonomy of Mixed Reality Visual Displays*, IEICE Trans, Information Systems, vol. E77-D, no.12, 1994.

[27] Minsker M., *Augmented Reality Is a Real Marketing Tool*, CRM Magazine, Vol. 18, No.2, 2014.

[28] Nevatia Y., Chintamani K., Meyer T., Blum T., Runge A., Fritz N., Computer Aided Medical Diagnosis And Surgery System: Towards automated medical diagnosis for long term space missions, ESA, 2011.

[29] Perey C., *Print and publishing and the future of Augmented Reality*, Information Services & Use 31, 2011.

[30] Prochazka D., Koubek T., *Augmented Reality Implementation Methods in Mainstream Applications*, arXiv preprint arXiv:1106.5569, 2011.

[31] Ribo M., Lang P., Ganster H., Brandner M., Stock C., i Pinz A., *Hybrid Tracking for Outdoor Augmented Reality Applications*, IEEE, Computer Graphics and Applications, 2002.

[32] Rombout L. E., Berkel A., Zakas L., *Interactive print – Developing with Layar Vision*, 2014.

[33] Sielhorst T., Feuerstein M., Navab N., *Advanced Medical Displays: A Literature Review of Augmented Reality*, Journal of display technology, VOL. 4, NO. 4, 2008.

[34] Stoyanova J., Gonçalves R., Brito P. Q., Coelho A., *Real-time Augmented Reality Pemo Platform for Exploring Consumer Emotional Responses with Shopping Applications*, International Journal of Online Engineering, Special No. 8, 2013.

[35] Stratton G. M., *Some preliminary experiments on vision without inversion of the retinal image, Psychological Review 3*, University of California, 1896.

[36] Sutherland I., *A Head-Mounted Three Dimensional Display*, The University of Utah, 1968.

[37] Sutherland I., *The Ultimate Display*, 1965.

[38] Tolliver-Nigro H., *Making the Most of the Mix of Mobile Marketing and Print,* The Seybold Report: Volume 11, Number 12, 2011.

[39] Trubow M., *Augmented reality marketing strategies: the how to guide for marketers*, Hidden Creative Ltd., 2011.

[40] Tseng W. K., Guo J. L., Liu C. P., *A comparison of Great Circle, Great Ellipse, and Geodesic Sailing*, Journal of Marine Science and Technology, Vol. 21, No. 3, 2013.

[41] Vallino J. R., *Interactive Augmented Reality*, University of Rochester, New York, 1998.

[42] Wasko C., *What teachers need to know about augmented reality enhanced learning environments*, Vol. 57, Number 4, 2013.

[43] Yuen S., Yaoyuneyong G., Johnson E., *Augmented reality: An overview and five directions for AR in education. Journal of Educational Technology Development and Exchange*, 2011.

ONLINE IZVORI

[1] *About Total Immersion*, Total Immersion, http://www.t-immersion.com/about-us/company-fact-sheet (4. Kolovoza 2018.)

[2] *About us*, Early-Adopter, https://early-adopter.com/about-us/ (27. Srpnja 2018.)

[3] Alvarez E., *Epic Games shows the potential of high-end augmented reality*, Engadget, 2017., https://www.engadget.com/2017/03/01/epic-games-project-raven-augmented-reality/ (14. Srpnja 2018.)

[4] *ARCore Overview*, Google Inc., https://developers.google.com/ar/discover/ (24. Kolovoza 2018.)

[5] *Augmented Reality bridges the gap between the digital and physical world*, Total Immersion, http://www.t-immersion.com/augmented-reality (16. Srpnja 2018.)

[6] *Augmented Reality promises astronauts instant medical knowhow*, ESA Int., 2012., http://www.esa.int/Our_Activities/Space_Engineering_Technology/Augmented_reality_promises_astronauts_instant_medical_knowhow (7. Kolovoza 2018.)

[7] *Behind the Scenes of Augmented Esquire*, Esquire, 2009., https://www.esquire.com/news-politics/news/g371/augmented-reality-technology-110909/ (7. Kolovoza 2018.)

[8] Bichlmeier C., *The CAMDASS Project – AR Mission to Space*, 2011., http://medicalaugmentedreality.com/2011/12/the-camdass-project-ar-mission-to-space/ (6. Kolovoza 2018.)

[9] *CAMDASS testing in Brussels*, ESA Int, 2012., http://m.esa.int/spaceinimages/Images/2012/01/CAMDASS_testing_in_Brussels (7. Kolovoza 2018.)

[10] Cao P., *ARKit 1.5 examples show how vertical surface mapping upgrades augmented reality in iOS 11.3*, 9To5Mac, 2018., https://9to5mac.com/2018/02/01/arkit-1-5-improvements-ios-11-3/ (24. Kolovoza 2018.)

[11] Churchill D., Lu J., Chiu T. K. F., Fox B., *Mobile Learning Design: Theories and Application*, 2015, https://books.google.hr/books?id=GFhECwAAQBAJ&pg (8. Kolovoza 2018.)

[12] Crawford S., *How Microsoft Kinect Works*, How Stuff Works, http://electronics.howstuffworks.com/microsoft-kinect2.htm (15. Kolovoza 2018.)

[13] *Custom Vision: Visual Intelligence Made Easy*, Microsoft Corporation, https://www.customvision.ai/ (17. Kolovoza 2018.)

[14] *Developer: Xcode*, Apple Inc., https://developer.apple.com/xcode/ (15. Kolovoza 2018.)

[15] *FLARToolKit*, Virtual Worldlets Network, http://www.virtualworldlets.net/Resources/Hosted/Resource.php?Name=FLARToolkit (1. Srpnja 2018.)

[16] *Google Maps Platform: Place Search*, Google Inc., https://developers.google.com/places/web-service/search (27. Kolovoza 2018.)

[17] *Google Maps Platform: Place Types*, Google Inc., https://developers.google.com/places/web-service/supported_types (27. Kolovoza 2018.)

[18] Hudson P., *How to round only specific corners using maskedCorners*, Hacking with Swift, 2018., https://www.hackingwithswift.com/example-code/calayer/how-to-round-only-specific-corners-using-maskedcorners (28. Kolovoza 2018.)

[19] Hudson P., *How to scale, stretch, move and rotate UIViews UsingCGAffineTransform*, Hacking with Swift, 2016., https://www.hackingwithswift.com/example-code/uikit/how-to-scale-stretch-move-and-rotate-uiviews-using-cgaffinetransform (28. Kolovoza 2018.)

[20] Huis D., *HDAugmentedReality*, GitHub Inc., https://github.com/DanijelHuis/HDAugmentedReality (27. Kolovoza 2018.)

[21] *Introducing ARKit: Augmented Reality for iOS*, Apple Inc., https://developer.apple.com/arkit/ (5. Kolovoza 2018.)

[22] Kettle M., *ABC Experiments with New Techniques to Promote FlashForward*, 2009., https://www.tvovermind.com/tv-news/abc-experiments-with-new-techniques-to-promote-flashforward (9. Kolovoza 2018.)

[23] *Kinect Hardware*, Microsoft Corporation, https://developer.microsoft.com/en-us/windows/kinect/hardware (15. Kolovoza 2018.)

[24] *Kinect tools and resources*, Microsoft Corporation, https://developer.microsoft.com/en-us/windows/kinect/tools (15. Kolovoza 2018.)

[25] Klein G., Murray D., *Parallel Tracking and Mapping for Small AR Workspaces*, University of Oxford, 2007., http://www.robots.ox.ac.uk/~gk/publications/KleinMurray2007ISMAR.pdf (23. Srpnja 2018.)

[26] Klein G., *Visual Tracking for Augmented Reality*, University of Cambridge, 2006., http://www.robots.ox.ac.uk/~gk/publications/Klein2006Thesis.pdf (23. Srpnja 2018.)

[27] Lamb P., *The world's most widely used tracking library for augmented reality*, https://www.hitl.washington.edu/artoolkit/ (28. Lipnja 2018.)

[28] Lardinois F., *Apple Launches Swift, A New Programming Language For Writing iOS And OS X Apps*, TechCrunch, 2014., https://techcrunch.com/2014/06/02/apple-launches-swift-a-new-programming-language-for-writing-ios-and-os-x-apps/ (17. Kolovoza. 2018).

[29] Lavaras N., *iOptik augmented reality contact lens prototype to be unveiled at CES*, New Atlas, https://newatlas.com/ioptik-ar-contact-lens-ces/30310/ (7. Kolovoza 2018.)

[30] *Layar Augmented Reality browser*, Layar, http://www.layar.com/ (6. Travnja 2018.)

[31] *Mathf.Atan2*, Unity 3D, https://docs.unity3d.com/ScriptReference/Mathf.Atan2.html (14. Kolovoza 2018.)

[32] McKesson J. L., *Learning Modern 3D Graphics Programming: Blinn-Phong Model*, GitHub Inc., 2012., https://paroj.github.io/gltut/Illumination/Tut11%20BlinnPhong%20Model.html (26. Kolovoza 2018.)

[33] Moynihan T., *It's a Good Thing the F-35's $400K Helmet Is Stupid Cool*, Wired, 2016, https://www.wired.com/2016/06/course-f-35-comes-400000-augmented-reality-helmet/ (7. Kolovoza 2018.)

[34] *NyARToolKit*, Virtual Worldlets Network, http://www.virtualworldlets.net/Resources/Hosted/Resource.php?Name=NyARToolkit (1. Srpnja 2018.)

[35] *One glowing moment*, Sports Business Journal, 2017., http://www.sportsbusinessdaily.com/Journal/Issues/2017/01/23/NHL-at-100/Glowing-puck.aspx (14. Kolovoza 2018.)

[36] Pentenrieder K., Bade C., Doil F., Meier P., *Augmented reality-based factory planning - an application tailored to industrial needs*, 2007., http://dx.doi.org/10.1109/ISMAR.2007.4538822

[37] Puiu T., *The Difference between Virtual and Augmented Reality*, ZME Science, 2017., http://www.zmescience.com/other/did-you-know/difference-virtual-augmented-reality/ (3. Kolovoza 2018.)

[38] *Qualcomm Announces Availability of Augmented Reality SDK*, Qualcomm, 2010. https://www.qualcomm.com/news/releases/2010/10/04/qualcomm-announces-availability-augmented-reality-sdk (5. kolovoza 2018.)

[39] *Quickstart for iOS*, Google Inc., https://developers.google.com/ar/develop/unity/quickstart-ios (24. Kolovoza 2018.)

[40] *RoomAlive*, Microsoft Corporation, https://www.microsoft.com/en-us/research/project/roomalive/?from=http%3A%2F%2Fresearch.microsoft.com%2Fen-us%2Fprojects%2Froomalive%2F# (15. Kolovoza 2018.)

[41] Rouse M., *RGB (red, green, and blue)*, WhatIs, 2005., https://whatis.techtarget.com/definition/RGB-red-green-and-blue (26. Kolovoza 2018.)

[42] Silver S., *iOS 11.3 with battery improvements, ARKit 1.5, HomeKit authentication now available for iPhone*, Apple Insider, https://appleinsider.com/articles/18/03/29/ios-113-with-battery-improvements-arkit-refinements-now-available-for-iphone-ipad-and-ipod-touch (24. Kolovoza 2018.)

[43] Symmes D., *The Chopper*, 2006., https://archive.is/20110707063258/http://www.3dmovingpictures.com/chopper.html (14. Kolovoza 2018.)

[44] *Teachable Machine*, Google Inc., https://teachablemachine.withgoogle.com/ (17. Kolovoza 2018.)

[45] *Units of Longitude and Latitude*, NAAP Astronomy Labs
http://astro.unl.edu/naap/motion1/tc_units.html
(26. Kolovoza 2018.)

[46] *Vuforia Documentation*, Unity 3D,
https://docs.unity3d.com/Manual/vuforia-sdk-overview.html
(24. Kolovoza 2018.)

[47] *Vuforia: About us*, LinkedIn,
https://www.linkedin.com/company/vuforia-a-ptc-technology/
(24. Kolovoza 2018.)

[48] *Working with Core ML Models*, Apple Inc.,
https://developer.apple.com/machine-learning/build-run-models/
(16. Kolovoza 2018.)

[49] Yuen S., *3D Augmented Reality Books*, WordPress, 2010.,
https://scyuen.wordpress.com/2010/11/19/3d-augmented-reality-books/
(3. Srpnja 2018.)

[50] *YUV Colorspace*, SoftPixel,
http://softpixel.com/~cwright/programming/colorspace/yuv/
(26. Kolovoza 2018.)

[51] Zhou F., Duh H.B.L., Billinghurst M., *Trends in augmented reality tracking,interaction and display*: *A review of ten years of ismar*, The HIT Lab NZ, University of Canterbury, New Zealand, 2008.
https://ir.canterbury.ac.nz/bitstream/handle/10092/2345/12613246_2008-Trend-inAugmentedRealityTrackingInteractionandDisplayAReviewofTenYearsofISMAR.pdf?sequence=1&isAllowed=y (15. Travnja 2018.)

[52] Zia Z., *What is Visual inertial odometry*, Quora, 2016.,
https://www.quora.com/What-is-visual-inertial-odometry-and-steps-involved-in-it
(5. Kolovoza 2018.)

POPIS SLIKA

http://www.esa.int/spaceinimages/Images/2012/01/CAMDASS_testing_in_Brussels (7. Kolovoza 2018.)

[11] **Slika 11.**, Elementi i prikaz *BARS* sustava, Livingston M. A., Rosenblum L. J., Julier S. J., Brown D., Baillot Y., Swan J. E. II, Gabbard J. L., Hix D., *An Augmented Reality System for Military Operations in Urban Terrain*, Proceedings of Interservice / Industry Training, Simulation & Education Conference (I/ITSEC), December 2-5, Orlando, Florida, 2002., str. 5.

[12] **Slika 12.** Koncept funkcioniranja *iOptik* sustava, Lavaras N., *iOptik augmented reality contact lens prototype to be unveiled at CES*, New Atlas, 2014., https://newatlas.com/ioptik-ar-contact-lens-ces/30310/ (7. Kolovoza 2018.)

[13] **Slika 13.**, Montiranje *F-35* kacige, Moynihan T., *It's a Good Thing the F-35's $400K Helmet Is Stupid Cool*, Wired, 2016., https://www.wired.com/2016/06/course-f-35-comes-400000-augmented-reality-helmet/ (7. Kolovoza 2018.)

[14] **Slika 14.**, Prvi prikaz dizajna interijera pomoću virtualnog namještaja u realnom okruženju popraćen emuliranjem umjetne rasvjete i sjena uz pomoć virtualnog objekta, Breen D. E., Whitaker R. T., Rose E., Tuceryan M., *Interactive Occlusion and Automatic Object Placement for Augmented Reality*, California Institute of Technology, Pasadena, 1996.

[15] **Slika 15.**, Koncept i prototip stolno baziranog sustava za proširenu stvarnost, Kato H., Billinghurst M., Poupyrev I., Imamoto K., Tachibana K., *Virtual Object Manupulation on a Table-Top AR Environment*, 2000.

[44] **Slika 16.**, Međusobna suradnja studenata i interakcija sa instruktorom u *Construct3D* okruženju, Kaufmann H., Schmalstieg D., *Mathematics and geometry education with collaborative augmented reality*, SIGGRAPH 2002 conference abstracts and applications, San Antonio, 2002.

[16] **Slika 17.**, Praktični prikaz sadržaja knjige *The Future is Wild: The Living Book*, *Augmented Reality T-Shirt and the Living Book from The Future is Wild*, YouTube, 2010., https://www.youtube.com/watch?v=JqUTWfTgoHM (9. Kolovoza 2018.)

[17] **Slika 18.**, David Granger, urednik *Esquire* magazina, demonstrira funkcionalnost proširene stvarnosti u kombinaciji sa računalom, *Editor David Granger demos the*

interactive magazine, YouTube, 2012., https://www.youtube.com/watch?v=LGwHQwgBzSI (9. Kolovoza 2018.)

[18] **Slika 19.**, Promotivni letak tvrtke *MINI* iz 2008. god. s elementima proširene stvarnosti, Strauss P., *Mini Augmented Reality Ads Hit Newstands*, Technabob, 2008., https://technabob.com/blog/2008/12/17/mini-augmented-reality-ads-hit-newstands/ (9. Kolovoza 2018.)

[19] **Slika 20.**, Izgled i prikaz sadržaja *Total Immersion* razglednice putem Web kamere, *Interactive Greeting Card*, Total Immersion, 2009., http://greetings.t-immersion.com/ (9. Kolovoza 2018.)

[20] **Slika 21.**, *Hockey puck* s integriranim odašiljačima i prikaz istog putem televizijskog prijenosa, *One glowing moment*, SportsBusiness Journal, 2017., http://www.sportsbusinessdaily.com/Journal/Issues/2017/01/23/NHL-at-100/Glowing-puck.aspx (14. Kolovoza 2018.)

[21] **Slika 22.**, 3D Naočale iz 1924. godine, Davis E., *See what 3D Glasses Looked Like Back in 1924.*, Fandango, 2014., https://www.fandango.com/movie-news/see-what-3d-glasses-looked-like-back-in-1924-748122 (14. Kolovoza 2018.)

[22] **Slika 23.**, *Mill Blackbird* (lijevo) sirova snimka, *Chevrolet Camaro ZL1* i *Chevrolet FNR* (desno) u kombinaciji sa *Unreal Engine* aplikacijom u realnom vremenu, Alvarez E., *Epic Games shows the potential of high-end augmented reality*, Engadget, 2017., https://www.engadget.com/2017/03/01/epic-games-project-raven-augmented-reality/#/ (14. Kolovoza 2018.)

[23] **Slika 24.**, *Nokia 6630* i *AR Tennis* aplikacija u primjeni, Billinghurst M., *AR Tennis*, University of Caterbury, 2006.,

[24] **Slika 25.**, *Microsoft Kinect* uređaj u kombinaciji sa *BenQ* projektorom, Warren T., *Microsoft's 'RoomAlive' transforms any room into a giant Xbox game*, The Verge, 2014., https://www.theverge.com/2014/10/5/6912979/microsoft-roomalive-research-projector-system (15. Kolovoza 2018.)

[25] **Slika 26.**, Tijek rada *ARToolKit* alata, Barbosa S., Afonso E., Junior L., Oliveira Adrande A., Cardoso A., *Virtual and Augmented Reality: A New Approach to Aid Users of Myoelectric Prostheses*, Intech, 2012., str. 415.

[26] **Slika 27.**, Praktični prikaz funkcionalnosti *FLARToolKit* biblioteke, Koyama T., *Introduction to FLARToolKit*, 2009., https://saqoo.sh/a/labs/FLARToolKit/Introduction-to-FLARToolKit.pdf (1. Srpnja 2018.)

[27] **Slika 28.**, *D'Fusion Studio* grafičko korisničko sučelje, *Total Immersion Releases Augmented Reality Platform Free To Developers,* Total Immersion, http://www.t-immersion.com/press-room/press-release/total-immersion-releases-augmented-reality-platform-free-developers (4. Kolovoza 2018.)

[28] **Slika 29.**, Dijagram toka podataka *AR SDK* u aplikacijskom okruženju, *Developing Android Augmented Reality Applications*, DeveloperIQ, 2014., http://developeriq.in/articles/2014/sep/25/developing-android-augmented-reality-applications/ (5. Kolovoza 2018.)

[29] **Slika 30.**, Demonstracija *Vuforia* aplikacije za *HoloLens HMD* putem *Unity* razvojnog alata, Dachis A., *Vuforia Brings a Discoverable Holographic World to the HoloLens*, Next Reality, 2016., https://hololens.reality.news/news/vuforia-brings-discoverable-holographic-world-hololens-0173089/ (24. Kolovoza 2018.)

[30] **Slika 31.**, Primjer funkcionalnosti aplikacije bazirane na *ARCore* platformi, Schenck S., *ARCore v1.2 gives devs new tools to create collaborative AR experiences*, Android Police, 2018., https://www.androidpolice.com/2018/05/11/arcore-v1-2-gives-devs-new-tools-create-collaborative-ar-experiences-apk-download/ (24. Kolovoza 2018.)

[31] **Slika 32.**, Test *ARKit* Example aplikacije na *iPad* uređaju, Graham L., Apple's new software is a game changer for augmented reality expets say, CNBC Tech, 2017., https://www.cnbc.com/2017/06/07/apple-arkit-software-game-changer-for-augmented-reality-ar-tech-say-experts.html (5. Kolovoza 2018.)

[32] **Slika 33.**, Odabir vrste aplikacije prilikom kreiranja novog projekta u *Xcode* sučelju, Izradio autor

[33] **Slika 34.**, *Xcode* izbornik priključenih uređaja za testiranje aplikacije, Izradio autor

[34] **Slika 35.**, Početni zaslon prilikom pokretanja aplikacije i postavljanje početne točke, Izradio autor

[35] **Slika 36.**, Odabir krajnje točke mjerenja i prikaz obiju točaka vezanih za objekt u prostoru, Izradio autor

[36] **Slika 37.**, *Vuforia Developer Portal*, Izradio autor

[37] **Slika 38.**, Proces izrade trodimenzionalnog modela Unipu logotipa, Izradio autor

[38] **Slika 39.**, Proces izrade aplikacije u *Unity 3D* softveru, Izradio autor

[39] **Slika 40.**, Prikaz procesa programiranja *unipuController.cs* skripte za upravljanje objekta u prostoru unutar *Visual Studio* aplikacije, Izradio autor

[40] **Slika 41.**, Proces pokretanja kreirane aplikacije putem *Xcode* sučelja, Izradio autor

[41] **Slika 42.**, Testiranje funkcionalnosti izrađene aplikacije, Izradio autor

[42] **Slika 43.**, Proces pohrane slika uz odgovarajuću etiketu, Izradio autor

[43] **Slika 44.**, Prikaz učinkovitosti kreiranog modela računalnog učenja, Izradio autor

[44] **Slika 45.**, Proces konverzije i eksportiranja kreiranog *Core ML* modela, Izradio autor

[45] **Slika 46.**, Proces implementiranja vlastitog *ARVision* modela unutar sintakse i pokretanje aplikacije na *iOS* uređaju putem *Xcode* sučelja, Izradio autor

[46] **Slika 47.**, Traka sa statistikama na temelju *SceneKit* biblioteke, Izradio autor

[47] **Slika 48.**, Prikaz funkcije prepoznavanja predmeta pomoću kreirane aplikacije, Izradio autor

[48] **Slika 49.**, Početni zaslon s prikazom trenutne lokacije (plavo) i najbližih točaka interesa (crveno), Izradio autor

[49] **Slika 50.**, Prikaz točaka interesa i njihove udaljenosti pomoću proširene stvarnosti, Izradio autor

[50] **Slika 51.**, Prikaz informacija dostupnih pri dodiru željene anotacije, Izradio autor

PRILOG 1.

U nastavku se nalazi kompletan kod *ViewController.swift* datoteke korišten u aplikaciji za mjerenje udaljenosti točaka u prostoru:

```
//
//  ViewController.swift
//  Measure App v5
//
//  Created by Ivan on 07/08/2018.
//  Copyright © 2018 Ivan. All rights reserved.
//

import UIKit
import SceneKit
import ARKit

class ViewController: UIViewController, ARSCNViewDelegate {

    @IBOutlet var sceneView: ARSCNView!

    var tocke: [SCNNode] = []

    var prikazMjerenja = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()

        prikazMjerenja.frame = CGRect(x: 0, y: 0, width:
view.frame.size.width, height: 100)

        prikazMjerenja.backgroundColor = .white

        prikazMjerenja.text = String(format: "Udaljenost: %.2f
cm")

        prikazMjerenja.textAlignment = .center

        view.addSubview(prikazMjerenja)

        sceneView.delegate = self

        sceneView.showsStatistics = true

        let dodirPrepoznavanje = UITapGestureRecognizer(target:
self, action: #selector(upravljanjeDodira))

        dodirPrepoznavanje.numberOfTapsRequired = 1

        sceneView.addGestureRecognizer(dodirPrepoznavanje)
    }
```

```
    @objc func upravljanjeDodira(sender:
UITapGestureRecognizer) {

        let location = sender.location(in: sceneView)

        let hitTest = sceneView.hitTest(location, types:
[ARHitTestResult.ResultType.featurePoint])

        guard let result = hitTest.last else { return }

        let transform = SCNMatrix4.init(result.worldTransform)

        let vector = SCNVector3Make(transform.m41,
transform.m42, transform.m43)

        let tocka = novaTocka(at: vector)

        if let first = tocke.first {

            tocke.append(tocka)
            prikazMjerenja.text = "Udaljenost:
\(tocka.udaljenost(to: first)) cm"

            if tocke.count > 2 {

                for tocka in tocke {

                    tocka.removeFromParentNode()
                }

                tocke = [tocke[2]]
            }

        } else {
            tocke.append(tocka)
        }

        for tocka in tocke {

            self.sceneView.scene.rootNode.addChildNode(tocka)
        }
    }

    func novaTocka(at position: SCNVector3) -> SCNNode {

        let tocka = SCNSphere(radius: 0.005)

        let node = SCNNode(geometry: tocka)

        node.position = position

        let material = SCNMaterial()

        material.diffuse.contents = UIColor.orange

        material.lightingModel = .blinn
```

```
            tocka.firstMaterial = material

            return node

        }

        override func viewWillAppear(_ animated: Bool) {
            super.viewWillAppear(animated)

            let postavke = ARWorldTrackingConfiguration()

            sceneView.session.run(postavke)
        }

        override func viewWillDisappear(_ animated: Bool) {
            super.viewWillDisappear(animated)

            sceneView.session.pause()
        }
    }


    extension SCNNode {

        func udaljenost(to odrediste: SCNNode) -> CGFloat {

            let dx = odrediste.position.x - position.x

            let dy = odrediste.position.y - position.y

            let dz = odrediste.position.z - position.z

            let metri = sqrt(dx*dx + dy*dy + dz*dz)

            return CGFloat(metri * 100)
        }
    }
```

PRILOG 2.

U nastavku se nalazi u cijelosti ispisan kod *ViewController.swift* datoteke unutar aplikacije za prepoznavanje predmeta u okolini:

```
//
//  ViewController.swift
//
//  AR Vision
//
//  Created by Ivan on 17/08/2018.
//  Copyright © 2018 Ivan. All rights reserved.
//

import UIKit
import SceneKit
import ARKit
import Vision

class ViewController: UIViewController, ARSCNViewDelegate {

    @IBOutlet var sceneView: ARSCNView!
    let textDubina : Float = 0.01
    var MLProcjena : String = "…"
    var vizijaZahtjev = [VNRequest]()
    let dispatchQueueML = DispatchQueue(label:
"DispatchQueueML")


    override func viewDidLoad() {
        super.viewDidLoad()

        sceneView.delegate = self

        sceneView.showsStatistics = true

        let scene = SCNScene()

        sceneView.scene = scene

        sceneView.autoenablesDefaultLighting = true

        let dodir = UITapGestureRecognizer(target: self,
action: #selector(self.upravljanjeDodirom(dodirPercepcija:)))
        view.addGestureRecognizer(dodir)


        guard let odabirModela = try? VNCoreMLModel(for:
Resnet50().model) else {
            fatalError("Nije moguće učitati model.")
        }
```

```
        let klasifikacijaZahtjev = VNCoreMLRequest(model:
odabirModela, completionHandler:
upravljacIzvrseneKlasifikacije)
        klasifikacijaZahtjev.imageCropAndScaleOption =
VNImageCropAndScaleOption.centerCrop

        vizijaZahtjev = [klasifikacijaZahtjev]

        loopCoreMLUpdate()
    }

    override func viewWillAppear(_ animirano: Bool) {
        super.viewWillAppear(animirano)

        let postavke = ARWorldTrackingConfiguration()
        postavke.planeDetection = .horizontal

        sceneView.session.run(postavke)
    }

    override func viewWillDisappear(_ animirano: Bool) {
        super.viewWillDisappear(animirano)

        sceneView.session.pause()
    }

        override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }


    func renderer(_ renderer: SCNSceneRenderer, updateAtTime
time: TimeInterval) {
        DispatchQueue.main.async {
        }
    }

    override var prefersStatusBarHidden : Bool {
        return true
    }


    @objc func upravljanjeDodirom(dodirPercepcija:
UITapGestureRecognizer) {
        let zaslonSredina : CGPoint = CGPoint(x:
self.sceneView.bounds.midX, y: self.sceneView.bounds.midY)

        let arHitTestResults : [ARHitTestResult] =
sceneView.hitTest(zaslonSredina, types: [.featurePoint])

        if let najbliziRezultat = arHitTestResults.first {
            let pretvorba : matrix_float4x4 =
najbliziRezultat.worldTransform
            let koordinate : SCNVector3 =
SCNVector3Make(pretvorba.columns.3.x, pretvorba.columns.3.y,
pretvorba.columns.3.z)
```

```
                let tocka : SCNNode = novi3DText(MLProcjena)
                sceneView.scene.rootNode.addChildNode(tocka)
                tocka.position = koordinate
            }
        }

        func novi3DText(_ text : String) -> SCNNode {
    To reduce chances of crashing; reduce number of polygons,
    letters, smoothness, etc.

            let natpisOgranicenje = SCNBillboardConstraint()
            natpisOgranicenje.freeAxes = SCNBillboardAxis.Y

            let Text3D = SCNText(string: text, extrusionDepth:
    CGFloat(textDubina))
            var font = UIFont(name: "EuphemiaUCAS", size: 0.15)
            font = font?.saOsobinama(osobine: .traitBold)
            Text3D.font = font
            Text3D.alignmentMode = kCAAlignmentCenter
            Text3D.firstMaterial?.diffuse.contents = UIColor.orange
            Text3D.firstMaterial?.specular.contents = UIColor.white
            Text3D.firstMaterial?.isDoubleSided = true
            Text3D.chamferRadius = CGFloat(textDubina)

            let (minBound, maxBound) = Text3D.boundingBox
            let Text3DNode = SCNNode(geometry: Text3D)

            Text3DNode.pivot = SCNMatrix4MakeTranslation(
    (maxBound.x - minBound.x)/2, minBound.y, textDubina/2)

            Text3DNode.scale = SCNVector3Make(0.2, 0.2, 0.2)

            let kugla = SCNSphere(radius: 0.005)
            kugla.firstMaterial?.diffuse.contents = UIColor.white
            let kuglaNode = SCNNode(geometry: kugla)

            let Text3DNodeRoditelj = SCNNode()
            Text3DNodeRoditelj.addChildNode(Text3DNode)
            Text3DNodeRoditelj.addChildNode(kuglaNode)
            Text3DNodeRoditelj.constraints = [natpisOgranicenje]

            return Text3DNodeRoditelj
        }


        func loopCoreMLUpdate() {

            dispatchQueueML.async {
                self.updateCoreML()
                self.loopCoreMLUpdate()
            }
        }
```

```
 func upravljacIzvrseneKlasifikacije (zahtjev: VNRequest,
greska: Error? ) {

        if greska != nil {
            print("Greška: " + (greska?.localizedDescription)!)
            return
        }
        guard let zapazanja = zahtjev.results else {
            print("Nema rezultata")
            return
        }

        let klasifikacije = zapazanja[0...1]
            .flatMap({ $0 as? VNClassificationObservation })
            .map({ "\($0.identifier) \(String(format:"- %.2f",
$0.confidence))" })
            .joined(separator: "\n")


        DispatchQueue.main.async {

            print(klasifikacije)
            print("--")

            var imeObjekta:String = "…"
            imeObjekta = klasifikacije.components(separatedBy:
"-")[0]
            imeObjekta = imeObjekta.components(separatedBy:
",")[0]
            self.MLProcjena = imeObjekta

        }
    }

    func updateCoreML() {
        let privremeniSpremnik : CVPixelBuffer? =
(sceneView.session.currentFrame?.capturedImage)
        if privremeniSpremnik == nil { return }
        let ciImage = CIImage(cvPixelBuffer:
privremeniSpremnik!)

        let upravljacZahtjevaSlike =
VNImageRequestHandler(ciImage: ciImage, options: [:])
VNImageRequestHandler(cgImage: cgImage!, orientation:
myOrientation, options: [:])

        do {
            try
upravljacZahtjevaSlike.perform(self.vizijaZahtjev)
        } catch {
            print (error)
        }
    }

}
```

```
 extension UIFont {
    func saOsobinama
(osobine:UIFontDescriptorSymbolicTraits...) -> UIFont {
        let descriptor =
self.fontDescriptor.withSymbolicTraits(UIFontDescriptorSymbolic
Traits(osobine))
        return UIFont(descriptor: descriptor!, size: 0)
    }
}
```