Primjena i implementacija tehnologije proširene stvarnosti na mobilnim platformama

Mrganić, Stjepan

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:092970

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-12-21**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

PRIMJENA I IMPLEMENTACIJA TEHNOLOGIJE PROŠIRENE STVARNOSTI NA MOBILNIM PLATFORMAMA

Završni rad

Stjepan Mrganić

Osijek, 2020.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 01.09.2020.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime studenta:	Stjepan Mrganić
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4107, 20.09.2019.
OIB studenta:	87632073223
Mentor:	Doc.dr.sc. Josip Balen
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Primjena i implementacija tehnologije proširene stvarnosti na mobilnim platformama
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	01.09.2020.
Datum potvrde ocjene Odbora:	09.09.2020.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 14.09.2020.

Ime i prezime studenta:	Stjepan Mrganić
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4107, 20.09.2019.
Turnitin podudaranje [%]:	6

Ovom izjavom izjavljujem da je rad pod nazivom**: Primjena i implementacija tehnologije proširene stvarnosti na mobilnim platformama**

izrađen pod vodstvom mentora Doc.dr.sc. Josip Balen

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Sadržaj

1. UVOD ... 1

1.1. Zadatak završnog rada ... 2

2. PROŠIRENA STVARNOST ... 3

2.1. Razlika između proširene i virtualne stvarnosti ... 3

2.2. Primjena proširene stvarnosti ... 5

2.3. Alati za razvoj AR aplikacija ... 8

3. IMPLEMENTACIJA PROŠIRENE STVARNOSTI U ANDROID APLIKACIJI ... 9

3.1. Postavljanje razvojnog okruženja ... 9

3.2. Aplikacijsko programsko sučelje Poly ... 11

3.3. Odabir modela i prikaz minijatura ... 16

3.4. Korisničko iskustvo ... 20

3.5. ARCore i Sceneform kompleti za razvoj aplikacija proširene stvarnosti ... 25

4. ZAKLJUČAK ... 34

LITERATURA ... 35

POPIS I OPIS UPOTRIJEBLJENIH KRATICA ... 38

SAŽETAK ... 39

ABSTRACT ... 40

ŽIVOTOPIS ... 41

PRILOZI ... 42

P.3.0. Izvorni kod aplikacije ... 42

P.3.1. JSON podatci ... 42

1. UVOD

Proširena i virtualna stvarnost su relativno nove i uzbudljive grane informacijske tehnologije koje su donedavno bile tek neostvarena ideja računalnih inženjera. Konstantni napredak, optimizacija razvojnih alata i sve složenije sklopovlje napokon omogućuju razvoj dosada nedostižnih aplikacija. Kulminacija nekoliko desetljeća dugog napretka može se primijetiti proučavanjem trenutno dostupne tehnologije. Današnje aplikacije upotrebom proširene stvarnosti poboljšavaju korisničko iskustvo i olakšavaju nekoć kompleksne zadatke.

Izrada aplikacije proširene stvarnosti zahtjeva znanja i primjenu tehnologija iz nekoliko različitih grana računalne znanosti. Kako je slučaj sa svim novitetima, nedostatak standardiziranog pristupa i edukativnog materijala, poput udžbenika, predstavljaju problem. Različiti kompleti za razvoj programa i dalje se natječu za prevlast na tržištu te zbog ekstremno brzih i učestalih promjena nije jasno tko će postaviti održiv industrijski standard. Unutar glavnog dijela završnog rada teorijski se proučavaju i praktično primjenjuju neke od trenutno dostupnih tehnologija.

Glavni problemi koje je potrebno riješiti su izrada ili pronalazak modela za prikaz korisniku, pozicioniranje korisnika koristeći dostupne senzore uređaja i rukovanje s postavljenim modelima. Pribavljanje modela riješeno je pomoću Poly aplikacijskog programskog sučelja. Kako bi se korisniku omogućila interakcija s prostorom u kojemu se nalazi potrebno je uspješno pronaći i stvoriti digitalnu reprezentaciju vertikalnih i horizontalnih površina. Sljedeće je potrebno ugraditi određeno prostorno razumijevanje, poput udaljenosti uređaja od prepreka. Problemi lokalizacije riješeni su primjenom ARCore kompleta za razvoj programa. Nakon prepoznavanja dostupnih površina potrebno je korisniku omogućiti postavljanje i rukovanje s 3D tj. trodimenzionalnim modelima. Njihovo iscrtavanje i cjelokupna obrada grafike riješena je koristeći Sceneform komplet za razvoj programa.

Završni rad navodi prednosti i potencijalne primjene proširene stvarnosti na probleme današnjice. Izrađena je jednostavna aplikacija koja prikazuje njene glavne koncepte, potkrepljene primjerima. Koristeći aplikaciju kao primjer dano je rješenje čestih problema programskih inženjera.

1.1. Zadatak završnog rada

Zadatak ovoga rada je objasniti pojmove proširene i virtualne stvarnosti, naglasiti razlike između njih i navesti postojeće primjere upotrebe proširene stvarnosti. Navode se i razlozi odabira pojedinih alata i njihovo korištenje unutar Android Studio integriranog razvojnog okruženja (engl. IDE – *Integrated development environment*). Konkretna izrada jednostavne aplikacije demonstrirana je korištenjem ARCore i Sceneform kompleta za razvoj programa (engl. SDK – *software development kit*) uz Poly aplikacijsko programsko sučelje (engl. API – *application programming interface*).

2. PROŠIRENA STVARNOST

2.1. Razlika između proširene i virtualne stvarnosti

Glavna značajka virtualne stvarnosti, za koju je kratica VR (engl. *Virtual reality*), je potpuni odmak od klasične definicije stvarnog prostora. Iako nije ograničena na sljedeće pojmove, ona uključuje interakciju s predmetima, audio-vizualne podražaje, standardno prihvaćanje omjera trodimenzionalnosti i pojam lokaliziranosti. Najbolja primjena virtualne stvarnosti je potpuno potiskivanje vanjskog svijeta i njegova zamjena računalno generiranim objektima. VR je odličan alat za stvaranje potpuno novih iskustava ili prilagodbu poznatih. Uobičajeni elementi VR iskustva su „naočale" s ugrađenim zaslonom i slušalicama, kontrolori za interakciju s objektima, često po jedan za svaku ruku i bazne stanice koje omogućuju korisnikovu detekciju u prostoru. Ovakav sustav je karakteristično stacionaran, ali učinkovit.

Jednako je primjenjiv za obrazovne svrhe, kao i za svrhe razonode. Postoji potreba za primjenom aplikacija virtualne stvarnosti koje pomažu učenicima, profesorima i studentima. Studenti medicine mogu u trodimenzionalnom prostoru proučavati anatomiju čovjeka, poput vena, kapilara, plućnog tkiva ili građe srca. Slični prikazi su u prošlosti bili prostorno ograničeni na dvije dimenzije (slika unutar udžbenika) ili su bili vremenski ograničeni na trajanje fakultetskih laboratorijskih vježbi. Unutar medicinske struke VR se primarno danas koristi na području kirurgije [1], ali vjerojatno je očekivati i širenje u druga područja. Prema [2] VR je potrebno razvijati kao integralni dio procesa obučavanja i edukacije. Potrebno ga je primijeniti uz već postojeće tradicionalne alate kako bi se nedostatci jednoga pristupa mogli nadopuniti s prednostima drugoga. VR nije postao općeprihvaćena tehnologija jedino na obrazovnoj domeni. Također je vrlo popularan u industriji videoigara. U ožujku 2020. godine, javnosti je postala dostupna, kritički cijenjena videoigra Half-Life: Alyx, koja se u potpunosti temelji na primjeni VR tehnologije [3]. Igrač je potpuno prožet unutarnjim osjećajem perspektive prve osobe, koja nikada nije imala toliku snagu i utjecaj.

Proširena stvarnost, čija je kratica AR (engl. *Augmented reality*), ima sličan pristup, ali ponešto drugačiju izvedbu. AR kao svoj osnovni princip ima nadogradnju već postojeće stvarnosti, odnosno prostora. Cilj VR tehnologije je zamijeniti, u što većem postotku, stvarni svijet, dok je cilj AR tehnologije nadograditi postojeći prostor korisnim informacijama. Spomenuto se najčešće postiže postavljenjem 3D objekata korištenjem kamere mobilnog uređaja. Takav pristup ima određene

prednosti, ali i nedostatke. Jedna od prednosti je mobilnost. Pošto ne postoji ograničenje baznim stanicama, koje pomoću lasera odrađuju korisnikov položaj, na raspolaganju je mnogo veći prostor. Druga prednost je veća dostupnost zbog pristupačnije cijene i lakšeg upoznavanja s cjelokupnim procesom. Cijena iskustva je vrlo bitna prepreka. Kako bi se ostvarilo ugodno VR iskustvo, trenutno je potrebno uložiti nešto više od osam tisuća kuna u dodatke [4]. Osim toga, potrebno je posjedovati dovoljno djelotvorno računalo za obradu sve potrebne grafike virtualnog okruženja. Stavka je to koja također može koštati jednako, ako ne i više nego dodatci. Brana pristupa AR iskustvu je posjedovanje podržanog mobilnog uređaja, kojim se korisnici lakše mogu upustiti u istraživanje AR tehnologije. Interakcija s aplikacijama proširene stvarnosti u dobrom dijelu je slična interakciji s klasičnim mobilnim aplikacijama. Modeli se postavljaju dodirom ekrana, može ih se rotirati, povećati, smanjiti i premjestiti na drugu poziciju koristeći već poznate geste na dodirnom zaslonu te korisnici ne moraju učiti nove i nepoznate kontrole unutar virtualnog okruženja.

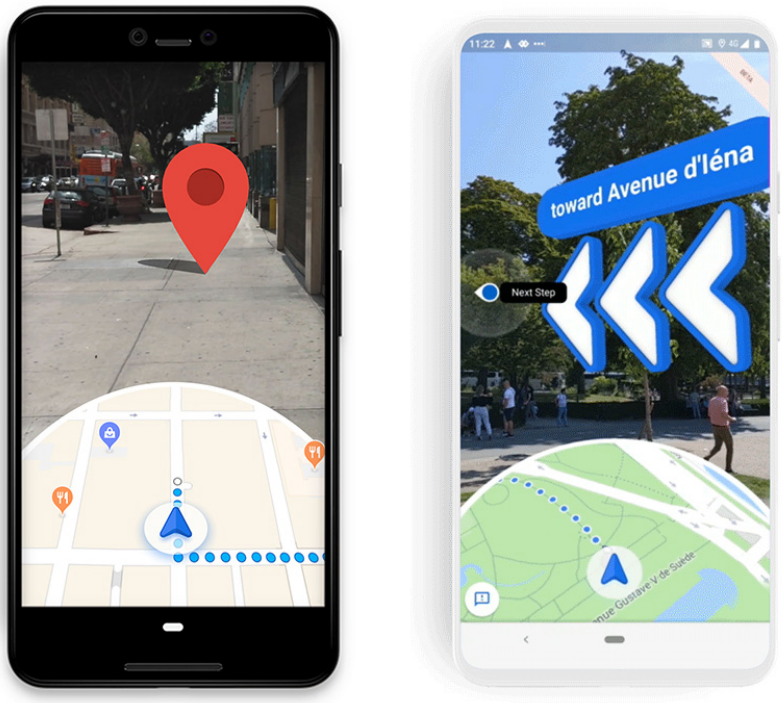
Lokalizacija, tj. određivanje položaja s obzirom na rotaciju, povišenje i lokaciju korisnika u usporedbi s drugim objektima u prostoru, također je područje različitosti. Ukoliko korisnik gleda zaslon uređaja, vidi prikaz slike stražnje kamere, i dalje posjeduje određenu količinu trenutnog prostornog razumijevanja. Relativno je ograničen vidnim poljem optike mobitela, koje je zasigurno uže od vidnog polja ljudskog oka. Korisnik koji, nasuprot tome, nosi naočale za virtualnu stvarnost, uopće nije svjestan fizičkog prostora u kojemu se nalazi, već jedino shvaća virtualne granice. Prilikom korištenja VR tehnologije velika je vjerojatnost da dođe do sudara s objektima u neposrednoj blizini ili čak s drugim ljudima. Prema [5] istraživači trenutno rade na algoritmima usmjeravanja koji bi mogli pomoći u istraživanju većih virtualnih prostora unutar ograničenja manjih fizičkih prostora. Dodatna prednost AR tehnologije je mogućnost dijeljenja iskustva s drugim korisnicima. Koristeći ARCore komplet za razvoj programa, moguće je iskoristiti značajku lokalizacije 3D sadržaja u oblaku (engl. *Cloud anchors*) [6] i dijeljenja istog u realnom vremenu. Korisnici mogu surađivati u kreiranju AR iskustva. Objekti koje jedan korisnik postavi u zajednički prostor će biti vidljivi i drugima. Jednako tako oboje će moći manipulirati objektima, neovisno o tome tko ih je postavio.

Nedostatak proširene stvarnosti naspram virtualne je preciznost u postavljanju modela i ograničeno prostorno shvaćanje. Moglo bi se reći da su to nužni kompromisi koji omogućuju povećanu mobilnost. Jasno je da dolazi do gubitka preciznosti, ukoliko se ne koriste laserski signali baznih stanica za praćenje korisnika. Senzori dostupni unutar mobilnog uređaja, iako manje precizni,

u međusobnoj kombinaciji mogu donekle nadoknaditi nedostatak preciznosti. Signali iz uređaja, poput akcelerometra, žiroskopa, kompasa, kamere i sustava globalnog pozicioniranja (engl. GPS – *Global positioning system*) moraju sinkronizirano raditi kako bi se ostvario zadovoljavajući rezultat.

2.2. Primjena proširene stvarnosti

Google karte se oslanjaju na sve dostupne senzore kako bi uspješno lokalizirali korisnike, osobito u područjima gdje GPS signal nije dovoljno pouzdan. Prema [7] velegradovi čije su ulice okružene visokim staklenim zgradama su odličan primjer. Fasade uzrokuju smetnje u prijenosu signala između satelita i prijemnika, stoga je gotovo nemoguće točno odrediti položaj uređaja koristeći samo GPS. Kombinacijom nekoliko različitih tehnologija prezentira se novi način navigacije. Nakon unošenja odredišta odabire se navigacija pomoću proširene stvarnosti. Aplikacija lokalizira korisnika koristeći sve prije navedene senzore i dodatno poboljšava točnost primjenjujući proces prepoznavanja ključnih točaka arhitekture opisan u [8], koristeći podatke kamere uspoređene sa bazom dostupnih snimki izgleda ulice. Slika 2.1. demonstrira način prikaza 3D putokaza na zaslonu uređaja. Korisnici kretanjem prema odredištu ispred sebe vide smjernice s informacijama pozicionirane na točni put kojim se trebaju kretati. Taj način navigacije trenutno je dostupan za testiranje svim korisnicima s podržanim uređajima.



Slika 2.1. Zasloni AR navigacije unutar aplikacije Google karte. [9] [10]

Google potiče inženjere da traže primjene proširene stvarnosti koje mogu unaprijediti već postojeća iskustva. Često se nova tehnologija koristi samo zato što je popularna; bez imalo razmišljanja poboljšava li postojeće stanje ili način izvedbe. Potrebno je pravilno odmjeriti koje alate upotrijebiti za rješavanje definiranog problema, a ne nužno koristiti one koji su najzanimljiviji ili najpoznatiji. Kao što [11] navodi, programski inženjeri trebaju biti upućeni u aktualne trendove unutar domene svoga posla, ali i unutar cjelokupne domene industrije.

Druga zanimljiva i korisna primjena AR tehnologije je već godinama dostupna korisnicima. Aplikacija Google prevoditelj, zanimljivo, ne dolazi predinstalirana na novim Android mobilnim uređajima, već ju je potrebno zasebno učitati s interneta. Neovisno o tome, radi se o jednoj od najboljih, danas dostupnih, aplikacija za prevođenje. Svoju prednost i u prošlosti je gradila na novim tehnologijama, poput primjene strojnog učenja na prevođenje teksta i ljudskog govora [12]. Taj trend prati se i danas. Unutar aplikacije moguće je pritiskom na tipku „Fotoaparat“ otvoriti kameru mobilnog uređaja i usmjeriti ju prema tekstu napisanom bilo kojim od osamdeset i osam podržanih jezika [13]. Aplikacija će prepoznati tekst, automatski odrediti o kojem je jeziku riječ i pomoću proširene stvarnosti ga zamijeniti našim preferiranim. Cijelom procesu potrebno je svega nekoliko sekundi za veće rečenične konstrukcije, a ukoliko se radi o prevođenju uličnih znakova, rezultat je gotovo trenutačan. Ova odlična primjena olakšava snalaženje u stranim zemljama i čitanje uputa ili priručnika koji kao jedini ponuđeni jezik imaju, npr. kineski. Iako takav pristup nije idealan, puno je bolji od alternative.

Slika 2.2. na sljedećoj stranici odlično dočarava prednosti AR iskustva. Potrebno je naglasiti da krajnji rezultat možda djeluje trivijalno, ali, ukoliko se detaljnije prouči, može se zaključiti da veliki broj tehnologija mora sinkronizirano djelovati. Prvo, potrebno je dekodirati signal kamere. Koristeći optičko prepoznavanje znakova, tekst pretvoriti u oblik koji se može obraditi. Nakon toga, potrebno je tekst provesti kroz program za detekciju jezika; tek tada je određen izvorišni format. Prikupljeni tekst potrebno je prevesti na odredišni jezik i naposlijetku se dolazi do dijela aplikacije koji radi s proširenom stvarnosti.

Aplikacija mora moći prepoznati površinu koja predstavlja znak. Prikazani slučaj je jedan od lakših za prepoznavanje. Znak po kontrastu i boji uvelike odudara od svoje okoline. Relativno je jednostavno uočiti njegove granice. Potrebno je razlučiti pozadinsku boju znaka i boju teksta kako bi AR prikaz odavao dojam pripadanja. Ukoliko se AR prijevod na površinu znaka pozicionira koristeći

crni kvadrat s bijelim tekstom, nisu uistinu praćeni ciljevi AR iskustva. Pošto je jedan od njih iluzija stvarne prisutnosti AR objekata, potrebno je uskladiti pozadinu i tekst objekta sa znakom. U suprotnom, puno jednostavniji pristup bio bi korisniku pokazati prevedeni tekst na zaslonu uređaja, a ne kroz pregled kamere. Na samome kraju, potrebno je objekt kvadrata blago transformirati, rotirati i translatirati kako ne bi izlazio iz granica znaka i na sličan način prekinuo iluziju. Nužno je kontinuirano pratiti ključne točke stvarnog objekta i ažurirati poziciju postavljenog virtualnog modela u stvarnom vremenu. Tako se osigurava uniformnost iskustva u slučaju da korisnik odluči promijeniti svoju perspektivu.

Slika 2.2. Zaslon aplikacije Google prevoditelj prilikom AR prevođenja. [14]

Ostvarivanje AR prijevoda zahtijeva prikaz relativno jednostavnih oblika i ažuriranje njihove pozicije. Konkretno se sada radi o 2D plohi koja je pozicionirana unutar 3D prostora. Jednostavnost modela dozvoljava aplikaciji „Prevoditelj“ kompatibilnost s velikim brojem uređaja. Time je ostvaren pristup kojemu bi sve aplikacije, čija svrha nije isključivo AR, trebale težiti. Krajnji program uistinu dobro prosuđuje dimenzije, uzme li se u obzir ograničen skup dostupnih ulaznih signala koji pomažu pri lokalizaciji.

2.3. Alati za razvoj AR aplikacija

Postoji nekoliko učinkovitih alata za razvoj složenih aplikacija specijaliziranih za proširenu stvarnost. Komplet alata i biblioteka za razvoj olakšava implementaciju velikom broju inženjera. AR aplikacije koriste složene procese za detekciju, praćenje i prikaz objekata. Rijetko tko posjeduje potrebna znanja i vještine potrebne za izradu tako složenog sustava. Besmisleno je iznova pisati kod koji obavlja istu funkcionalnost kao postojeća biblioteka, stoga i postoje pripremljeni alati, čijim se korištenjem znatno ubrzava razvojni proces. Svaki od alata ima svoje prednosti i nedostatke, zato je uputno prije početka projekta proučiti čije značajke najbolje odgovaraju rješavanju zadanog problema. Nasumičnim redoslijedom neki od najpopularnijih su:

- ARCore
- ARKit
- Vuforia
- Wikitude
- ARToolKit
- Kudan

Unutar okvira završnog rada koristi se ARCore, prvenstveno zbog odlično dostupne dokumentacije koda. Drugi razlog je cijena privilegije korištenja određenog alata. ARCore je dostupan za besplatno korištenje dok se, npr. Vuforia, Wikitude i Kudan naplaćuju. Svaki od njih ima drugačije načine pretplate i cjenovne razine, ali studentima je svakako najbolja besplatna opcija. ARToolKit je također besplatan, ali ima manje značajki u usporedbi s ARCore alatom. Broj inženjera koji koriste ARToolKit je vrlo malen, stoga, ukoliko dođe do problema prilikom razvoja, potencijalan skup osoba koje imaju sličan problem ili posjeduju volju za njegovim rješavanjem je također malen. ARCore, u vlasništvu Google LLC, redovito je održavan i ima vrlo aktivnu zajednicu programera, adekvatno razrađenu dokumentaciju i bogatu selekciju značajki spremnih za produkciju. Osim alata za Android Studio također nudi i alate za razvoj pomoću Unity i Unreal razvojnih okruženja. ARKit, u proizvodnji tvrtke Apple ostaje jedini nespomenut, a jednako je dobro razrađen u smislu dokumentacije i značajki. Dobar broj programera aktivno ga koristi, ali jedino podržava iOS platformu. S druge strane, ARCore, može se primijeniti na Android i iOS platformi. Pošto podržava veći broj uređaja, odabran je za korištenje.

3. IMPLEMENTACIJA PROŠIRENE STVARNOSTI U ANDROID APLIKACIJI

3.1. Postavljanje razvojnog okruženja

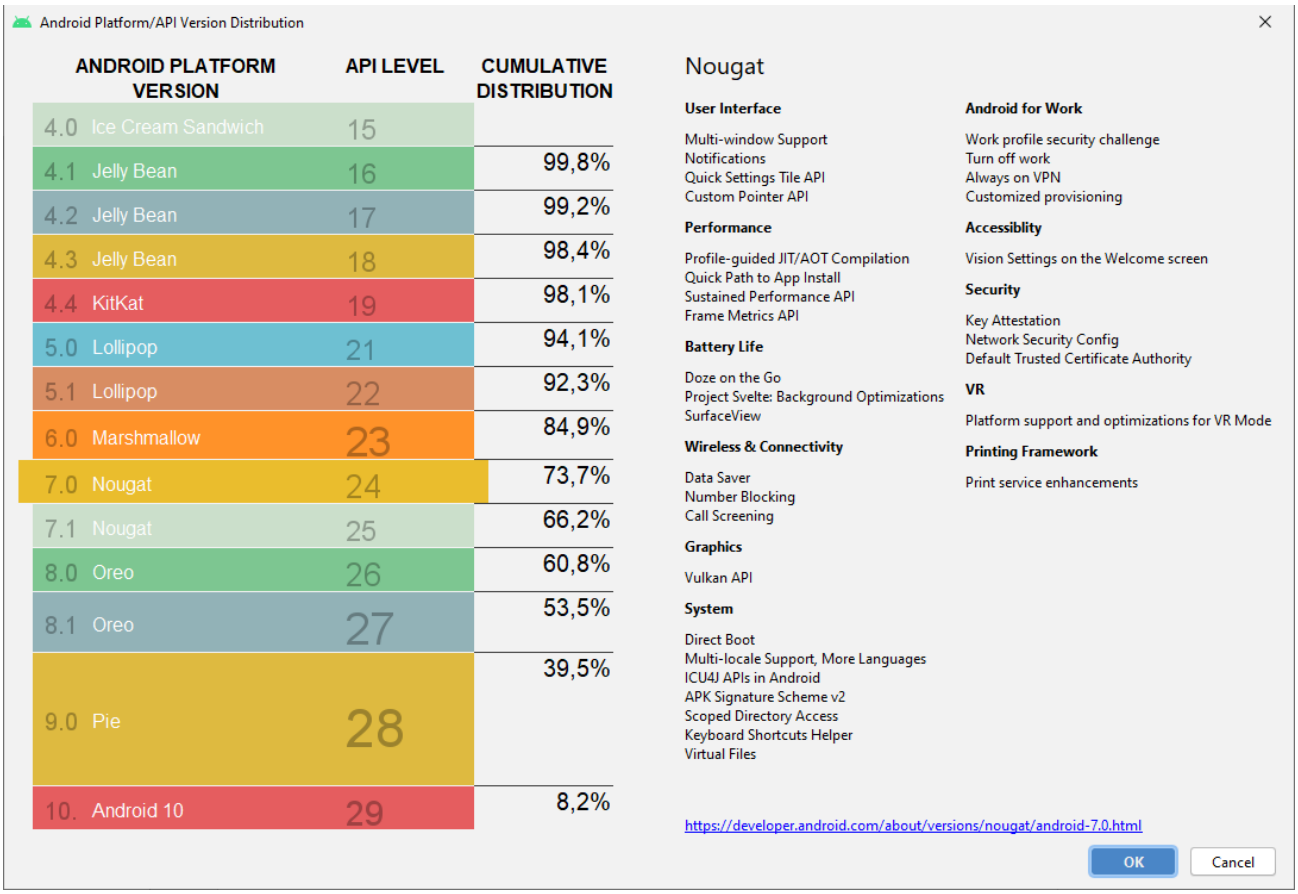
Nakon što su odabrani alati za izradu aplikacije, potrebno je postaviti razvojno okruženje. Pošto je rad fokusiran na izradu Android aplikacije prvo je potrebno preuzeti Android Studio. Najnovija verzija u trenutku pisanja završnog rada je 4.0. Izdana je u svibnju 2020. godine [15]. Pisanje završnog rada i testiranje odrađeno je na verziji 3.5.3 zbog nekompatibilnosti pojedinih značajki s novijim programom. Unatoč tome, završno kompajliranje i prevođenje programa odrađeno je uspješno i na najnovijoj verziji. Stoga se sadržaj završnog rada sigurno može primijeniti na verzije 4.0 i niže. Detaljno objašnjenje problema ponuđeno je unutar poglavlja ARCore i Sceneform.

Unutar Android Studio razvojnog okruženja stvoren je novi projekt s jednom praznom aktivnosti. Java je odabrana kao jezik pisanja programa. Originalno je zamišljena upotreba Kotlin programskog jezika, čije su osnove dobro objašnjene u [16]. Ubrzo se ispostavilo da je zadatak savladavanja novog jezika istovremeno s upotrebom mlade tehnologije ARCore dovoljno opširan pothvat za izradu minimalno dva završna rada. Većina primjera i dokumentacije za sada je napisana u Javi te još nije potpuno prevedena. Kotlin je svakako sposoban, uzbudljiv napredak na području programskih jezika te zaslužuje zasebnu obradu i pozornost.

Minimalna podržana verzija operacijskog sustava postavljena je na API 24: Android 7.0 (Nougat). Ovaj uvjet postavljen je od strane ARCore i Sceneform alata. Zbog velikog problema fragmentacije unutar Android sustava, odabir API razine 24 ograničava raspon podržanih uređaja na 73,7 posto. Slika 3.1. jasno prikazuje trenutno stanje fragmentacije, uzimajući u obzir sve dostupne platforme. Tijekom narednih nekoliko godina može se očekivati polagani rast tog postotka. Proizvođači često, nakon plasiranja uređaja u prodaju, prestaju s održavanjem njegovog operacijskog sustava. U najnesretnijim okolnostima to bude učinjeno već unutar prve godine životnog ciklusa. Kao posljedicu pred inženjere se stavlja zadatak odabira skupa uređaja koje mogu podržavati ili, gotovo nemoguć zadatak, pokrivanja cijelog spektra.

Google je vrlo svjestan tog problema i kao odgovor predstavlja modularni dizajn Android operacijskog sustava [17]. Postavlja uvjet podržavanja modularne nadogradnje programske podrške

svim proizvođačima koji žele pristupiti Google uslugama. Tim načinom dio odgovornosti održavanja preuzima s proizvođača na sebe. Kada cijeli sustav bude podržavao modularnu nadogradnju, problem fragmentacije bit će zadovoljavajuće razriješen. Nažalost, uvjet modularnog dizajna odnosi se samo na uređaje koji originalno dolaze opremljeni Android 10 operativnim sustavom te sigurno postoji pozamašno vremensko razdoblje u kojemu je i dalje nužno odabrati prioritete.



Slika 3.1. Stanje fragmentacije Android sustava. [18]

Nešto važniji pogled na cijelu situaciju je iz sigurnosne perspektive. Ukoliko samo 8,2 posto uređaja posjeduje ažuriranja s najnovijim zakrpama, postoji veliki sigurnosni rizik. Korisnici žele biti uvjereni u sigurnost svojih podataka i cjelokupne platforme. Vremensko razdoblje unutar kojega novi operacijski sustav postaje široko raširen je značajno manji kada se radi o iOS uređajima. Trenutno je 81 posto iOS uređaja ažurirano s najnovijim operacijskim sustavom [19]. Bitno je napomenuti da su najnovija izdanja oba sustava predstavljena u rujnu 2019. godine, stoga su imala jednako razdoblje da dospiju do uređaja krajnjih korisnika.

3.2. Aplikacijsko programsko sučelje Poly

Prvi problem prilikom izrade AR aplikacije je modeliranje, tj. izrada 3D objekata koji će se korisniku prikazati. Postoji nekoliko različitih rješenja tog problema. Prvi od njih je lasersko 3D skeniranje i računalno rekreiranje postojećih fizičkih modela. Cijene laserskih skenera kreću se od nekoliko tisuća do nekoliko stotina tisuća kuna, ovisno o kvaliteti proizvoda [20]. Razboritije je proučiti druge opcije. Sljedeći način je ručna izrada pomoću programa za izradu 3D grafike. Neki od najpoznatijih su:

- Maya
- Cinema 4D
- ZBrush
- Blender
- Autodesk 3ds Max
- Fusion 360
- Tilt Brush

Iako su svi izbori prihvatljivi, Blender je otvorenog izvornog koda (engl. *open source*), a smatra se i vodećim u industriji. Unutar aplikacije poželjno je prikazati više modela različitih veličina i oblika, a korisniku omogućiti isprobavanje modela koje sami prosude zanimljivima. Osobno dizajnirati kvalitetnu i raznovrsnu biblioteku elemenata smisleno je, ukoliko se stvara potpuno prilagođeno AR iskustvo. Poželjno je da modele dizajnira osoba s umjetničkom pozadinom i iskustvom. Vlastito kreiranje 3D modela, za potrebe završnog rada, nepotrebno bi povećalo složenost projekta i korisničko iskustvo svelo na nižu razinu.

Treći pristup je nešto jednostavniji. Kupiti komplet gotovih modela koji se mogu koristiti, učinkovito je, ali ponešto ograničeno. Prva prepreka je nedostatak raznovrsnosti koju se može uključiti u aplikaciju. Druga je činjenica da pakiranje nekoliko stotina modela unutar instalacijskog paketa nije optimiziran pristup. Odluči li se kasnije modele promijeniti, korisnici moraju ažurirati aplikaciju da dobiju pristup novima. Svakako, idealno je modele dohvatiti na uređaj tijekom izvođenja programa. Uzevši to u obzir, koristeći ovaj pristup, potrebno je implementirati vlastiti poslužitelj modela i bazu podataka unutar aplikacije koja čuva već preuzete resurse.

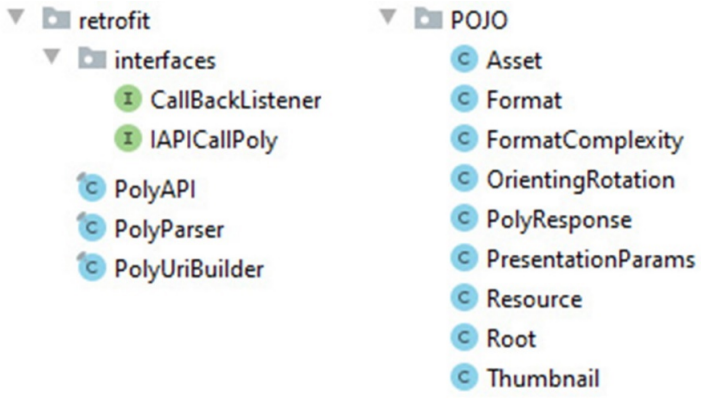
Četvrti izbor za rješavanje problema pribavljanja 3D modela je korištenje Poly aplikacijskog programskog sučelja. Google je također uočio probleme koji usporavaju razvoj AR aplikacija, stoga su odlučili inženjerima ponuditi online dostupnu, besplatnu biblioteku modela raznih vrsta [21]. Dodatno su ponudili RESTful API koji omogućava njihovo lako dohvaćanje i korištenje tijekom izvođenja. Dizajnerima su osigurali dodatke za najpopularnije programe izrade 3D grafika pomoću kojih mogu svoje kreacije izravno učitati u Poly. Primjenom Retrofit klijenta [22], Android aplikaciju lako se može povezati s mrežom i izvoditi HTTPS upite. Takav pristup najbolji je za jednostavnu demonstraciju proširene stvarnosti. Njegovom primjenom uspješno su razriješeni svi gore navedeni problemi.

Kada je poznat izvor 3D resursa, potrebno je dodati funkcionalnost spajanja s mrežom. Sustavi za izgradnju (engl. *build systems*) (Gradle i Maven) [23], omogućuju dodavanje javno dostupnih biblioteka u projekt koristeći malen broj linija koda. Unutar build.gradle datoteke potrebno je dodati ovisnosti nužne za Retrofit klijent.

```
implementation 'com.squareup.retrofit2:retrofit:2.9.0'
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
```

Programski kod 3.1. Ovisnosti Retrofit klijenta.

Nakon toga potrebno je stvoriti klase koje će služiti za dohvaćanje podataka i njihovo spremanje u memoriju. Sve klase koje služe isključivo za spremanje, mogu biti automatski generirane iz dolaznog JSON formata [24] koristeći Json2Pojo dodatak [25]. Prilog (P.3.1. JSON podatci) pokazuje primjer dolaznog formata iskorištenoga za generaciju Pojo (engl. *Plain old Java object*) klasa. Slika 3.2. prikazuje sva sučelja i klase potrebne za implementaciju Retrofit klijenta, organizirane unutar paketa zbog lakše preglednosti koda.



Slika 3.2. Sučelja i klase potrebne za API poziv.

Klasa PolyAPI napravljena je kako bi se funkcionalnost API poziva uklonila iz glavne aktivnosti. Ona enkapsulira svu potrebnu logiku za dohvaćanje informacija o modelima i nudi jedinstvenu točku mrežnog pristupa unutar cijele aplikacije. Programski kod 3.3. prikazuje metodu kojom se instancira Retrofit klijent. Ona koristi IAPICallPoly sučelje (Programski kod 3.2.), unutar kojega je moguće definirati sve potrebne HTTPS pozive. Npr. *GET, POST, PUT, DELETE* itd. PolyAPI je strukturiran tako da je jednim pozivom moguće dohvatiti listu elemenata koji odgovaraju pojmu za pretraživanje. Zato IAPICallPoly sadrži samo jedan *GET* poziv.

Retrofit za inicijalizaciju koristi oblikovni obrazac „Graditelj“ (engl. *builder*) koji je jasno objašnjen u literaturi [26]. U kratkim crtama *builder* odvaja stvaranje kompleksnog objekta od njegove reprezentacije, kako bi proces stvaranja mogao proizvesti više neovisnih reprezentacija.

```
public interface IAPICallPoly {
    @GET
    Call<PolyResponse> getListAssets(@Url Uri url);
}
```

Programski kod 3.2. GET poziv za odredišni URL.

```
private static IAPICallPoly apiInterface;
private static IAPICallPoly getApiInterface() {
    if (apiInterface == null) {
        Retrofit retrofit = new Retrofit.Builder()
                .baseUrl(BASE_API)
                .addConverterFactory(GsonConverterFactory.create())
                .build();
        apiInterface = retrofit.create(IAPICallPoly.class);
    }
    return apiInterface;
}
```

Programski kod 3.3. Instanciranje Retrofit klijenta.

Pošto nema svrhe stvarati objekte PolyAPI klase, definiran je prazan privatni konstruktor. Pošto je potrebno onemogućiti nasljeđivanje klase, ona je proglašena konačnom koristeći ključnu riječ *final.* Svi članovi klase označeni su kao statički, jednako kao i sve metode. Takvim oblikovanjem moguće je pristupati javnim metodama na razini klase bez instanciranja objekta, zato je nepotrebno stvaranje nekolicine objekata koji dijele jedinstvenu svrhu spajanja s mrežom.

Kako bi iz glavne aktivnosti bilo moguće obaviti API poziv, potrebno je izložiti javnu metodu koja to omogućuje. Programski kod 3.4. pokazuje nekoliko različitih inačica poziva. Polimorfizam dopušta definirati više metoda istoga imena, ali različitoga oblika. Svaka s različitim parametrom

poziva getPolyAPIResponse (Programski kod 3.5.), metodu koja je zadužena za upućivanje HTTPS zahtjeva na poslužitelj.

```
public static void callAPI() {
    getPolyAPIResponse(new PolyUriBuilder().build());
}
public static void callAPI(String keyword) {
    getPolyAPIResponse(
            new PolyUriBuilder()
                    .appendKeyword(keyword)
                    .build());
}
public static void callAPI(String keyword, int pageSize) {
    getPolyAPIResponse(
            new PolyUriBuilder()
                    .appendKeyword(keyword)
                    .appendpageSize(pageSize)
                    .build());
}
```

Programski kod 3.4. Javno izložene statičke metode.

```
private static void getPolyAPIResponse(Uri url) {
    Call<PolyResponse> PolyAPICall = getApiInterface().getListAssets(url);
    Callback<PolyResponse> callback = new Callback<PolyResponse>() {
        @Override
        public void onResponse(Call<PolyResponse> call, Response<PolyResponse> response) {
            if (response.isSuccessful()) {
                callBackListener.successfulResponse(response.body());
            }
        }
        @Override
        public void onFailure(Call<PolyResponse> call, Throwable t) {
            callBackListener.failedResponse((Exception) t);
        }
    };
    PolyAPICall.enqueue(callback);
}
```

Programski kod 3.5. Privatna metoda za mrežni poziv.

Metode callAPI koriste objekt „Graditelja“ (Programski kod 3.6.) koji na sebe preuzima odgovornosti formiranja URI (engl. *uniform resource identifier*) parametra, potrebnog metodi getPolyAPIResponse. „Graditelj“ pri pozivu konstruktora postavlja osnovne parametre upita. Jedan od njih je API ključ, koji zbog uspješne verifikacije upita mora biti postavljen, a generiran je pomoću stranice Google API Dashboard [27]. Postavljaju se dodatni filteri na upit, poput filtriranja prema formatu dostupnog modela i poput filtriranja pomoću ključne riječi *curated.* (*Curated* modeli su pozitivno ocijenjeni od strane Google tima za kontrolu kvalitete.) Moguće je i postaviti broj elemenata koje želimo dohvatiti jednim pozivom koristeći metodu appendPageSize.

```
public final class PolyUriBuilder {
    private final static String APIKey = "AIzaSyAKyQb2p4l9c4sJU_RahUXSpym2-E3Xjbs";
    private final static String TAG = "UriBuilder";
    private Uri.Builder uriBuilder;
    public PolyUriBuilder() {
        uriBuilder = new Uri.Builder()
                .scheme("https")
                .authority("poly.googleapis.com")
                .appendPath("v1")
                .appendPath("assets")
                .appendQueryParameter("key", APIKey)
                .appendQueryParameter("curated", Boolean.toString(true))
                .appendQueryParameter("format", "GLTF2");
    }
    public PolyUriBuilder appendKeyword(String keyword){
        if(keyword != null && !keyword.isEmpty()) {
            uriBuilder.appendQueryParameter("keywords", keyword);
        }
        return this;
    }
    public PolyUriBuilder appendPageSize(int pageSize){
        uriBuilder.appendQueryParameter("pageSize", String.valueOf(pageSize));
        return this;
    }
    public Uri build(){
        Log.d(TAG, "Used URL: " + this.toString());
        return uriBuilder.build();
    }
    @Override
    public String toString() {
        return uriBuilder.build().toString();
    }
}
```

Programski kod 3.6. URI graditelj.

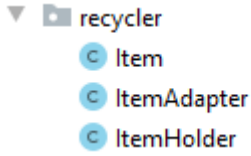
Pošto se mrežni pozivi odvijaju asinkrono, potrebno je implementirati povratni poziv (engl. *callback*) pomoću kojega se glavnu aktivnost obavještava o dostupnim podatcima ili pogreški. Tomu služi sučelje CallBackListener, na koje PolyAPI klasa drži referencu. Željenu funkcionalnost reakcije na promjenu stanja najbolje modelira oblikovni obrazac *observer* [26]. Radi njegove implementacije javno je izložena i metoda kojom se iz glavne aktivnosti CallBackListener može postaviti. U ovom slučaju klasa PolyAPI nakon primitka podataka, prosljeđuje ih prema glavnoj aktivnosti. Nakon što je aplikaciji omogućen pristup internetu unutar AndroidManifest.xml datoteke, stvoreni su svi preduvjeti za pribavljanje modela s interneta.

```
<uses-permission android:name="android.permission.INTERNET" />
```

Programski kod 3.7. Postavljanje pristupa internetu.

3.3. Odabir modela i prikaz minijatura

Pribavljene podatke pomoću API poziva potrebno je prikazati korisniku. Koristeći Retrofit jedino su uspješno dohvaćeni opisni podatci modela, poput njihovog imena, slike, lokacije itd., a nisu preuzete datoteke potrebne za prikaz 3D modela u okviru proširene stvarnosti. Njih će se morati preuzeti pomoću novog upita. Premda je moguće preuzimanje svih datoteka, ukoliko će korisnik odabrati samo nekolicinu, nije potrebno dohvaćanje dvadesetak modela. Zato su opisni podatci preuzeti s interneta prvo prikazani kao galerija mogućih izbora koristeći RecycleView. [28] RecycleView je Android komponenta koja omogućava programerima prikaz velikog skupa podataka koristeći relativno malo radne memorije, a najčešće su predočeni u obliku liste. Slika 3.3. prikazuje potrebne klase za njegovu implementaciju.



Slika 3.3. Klase potrebne za RecyclerView.

Item predstavlja jednostavni objekt koji služi za dohvaćanje i čuvanje samo nužnih podataka iz Pojo klasa. U ovom slučaju sadržavat će URI na datoteku 3D resursa, na 2D minijaturu i identifikator ID. ItemHolder predstavlja jedinstveni element koji se prikazuje. RecycleView instancira onoliko ItemHolder objekata koliko je potrebno za popunjavanje jednoga ekrana. Kada korisnik počne pretraživati listu, ne stvaraju se novi objekti, već recikliraju stari. Upravo tako se štedi na memoriji. ItemAdapter klasa brine o čuvanju liste podataka tipa Item koji se prikazuju na zaslonu i brine o instanciranju (Programski kod 3.1.) i recikliranju (Programski kod 3.9.) ItemHolder objekata.

```
public RecyclerView.ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    View layout = LayoutInflater
                  .from(parent.getContext())
                  .inflate(R.layout.thumbnail, parent, false);
    return new ItemHolder(layout, this);
}
```

Programski kod 3.8. Kreiranje objekta ItemHolder unutar klase *ItemAdapter.*

```
public void onBindViewHolder(@NonNull RecyclerView.ViewHolder holder, int position) {
    if (holder.getItemId() != position) {
        ((ItemHolder) holder).setItem(items.get(position));
        items.get(position).setHolder((ItemHolder) holder);
        ImageView iv = (ImageView) holder.itemView.findViewById(R.id.ivThumbnail);
        Picasso.get().load(items.get(position).getThumbnail()).into(iv);
    }
}
```

Programski kod 3.9. Recikliranje objekta ItemHolder.

Pošto je naslijeđena osnovna klasa RecycleView.Adapter, moraju se prepisati sve njene apstraktne metode. ItemAdapter iz glavne aktivnosti prima skup podataka i potpuno upravlja njegovim prikazivanjem. Glavna aktivnost ne mora pratiti broj svih elemenata; o tome se brine getItemCount. Klasi su dodane još dvije metode pomoću kojih se određuje i postavlja trenutno odabrani element.

```
private final List<Item> items;
private int selected;
public ItemAdapter(List<Item> items) {
    this.items = items;
    selected = -1;
}
public int getItemCount() { return items.size(); }
public Item getSelected() {
    if (selected >= 0) { return items.get(selected); }
    else return null;
}
public void setSelected(Item item) {
    if (item == null) { selected = -1; }
    else selected = items.indexOf(item);
}
```

Programski kod 3.10. Ostale metode klase ItemAdapter.

Klasa ItemHolder drži referencu na predani Item i ItemAdapter. Implementira onClickListener pomoću kojega se rukuje promjenom boje obruba odabrane minijature i ažurira ItemAdapter koristeći metodu setSelected (Programski kod 3.10.). Ovisno o usporedbi trenutno odabranog i samoga sebe, ItemHolder postavlja drugačije pozadine. Za dohvaćanje i učitavanje minijature koristi se javna biblioteka Picasso [29]. Naposlijetku, potrebno je dodati ovisnosti unutar build.gradle datoteke.

```
public void setItem(Item item) {
    this.item = item;
    ivThumbnail.setOnClickListener(this::onClick);
    if (item.equals(adapter.getSelected())) {
        ivThumbnail.setSelected(true);
        ivThumbnail.setBackgroundColor(SELECTED_VALUE);
    } else {
        ivThumbnail.setSelected(false);
        ivThumbnail.setBackgroundColor(DESELECTED_VALUE);
    }
}
private void onClick(View view) {
    Item selected = adapter.getSelected();
    if (!item.equals(selected)) {
        if (selected != null) {
            selected.getHolder().ivThumbnail.setBackgroundColor(DESELECTED_VALUE);
        }
        adapter.setSelected(item);
        ivThumbnail.setSelected(true);
        ivThumbnail.setBackgroundColor(SELECTED_VALUE);
    }
}
```

Programski kod 3.11. Promjena boje pozadine odabranog elementa.

```
implementation 'com.squareup.picasso:picasso:2.71828'
implementation "androidx.recyclerview:recyclerview:1.1.0"
```

Programski kod 3.12. Ovisnosti potrebne za korištenje biblioteka.

Nakon postavljanja struktura potrebnih za dohvaćanje i rukovanje podatcima potrebno je iz glavne aktivnosti poslati mrežni poziv i korisniku prikazati dobivene rezultate (Slika 3.4.). API poziv obavljamo koristeći točku pristupa enkapsuliranu unutar PolyAPI klase (Programski kod 3.13.). Pošto se poziv odvija asinkrono, PolyAPI koristi *observera* za povrat podataka u glavnu aktivnost. Stoga je potrebno kreirati jednog i postaviti ga na PolyAPI klasu. Unutar njegove definicije, ako su primljeni valjani podatci, oni se predaju u ItemAdapter i postavlja se RecycleView. U suprotnom se bilježi pogreška i ispisuje poruka korisniku. Kada je definiran *observer* moguće je uputiti poziv koristeći callAPI metodu.

```
CallBackListener onCallBack = new CallBackListener() {
    @Override
    public void successfulResponse(PolyResponse polyResponse) {
        if (polyResponse.isEmpty()) {
            Log.d(TAG, "No models for that keyword");
            snackBarHelper.showTimedMessage(
                    MainActivity.this,
                    getString(R.string.nothingForKeyword));
        } else {
            List<Item> items = PolyParser.parseListAssets(polyResponse);
            adapter = new ItemAdapter(items);
            adapter.setSelected(items.get(0));
            recyclerView.setAdapter(adapter);
        }
    }
    @Override
    public void failedResponse(Exception ex) {
        Log.d(TAG, "Retrofit failed");
        if (ex != null) ex.printStackTrace();
        snackBarHelper.showTimedMessage(
                        MainActivity.this,
                        getString(R.string.networkError));
    }
};
PolyAPI.setCallBackListener(onCallBack);
PolyAPI.callAPI();
```

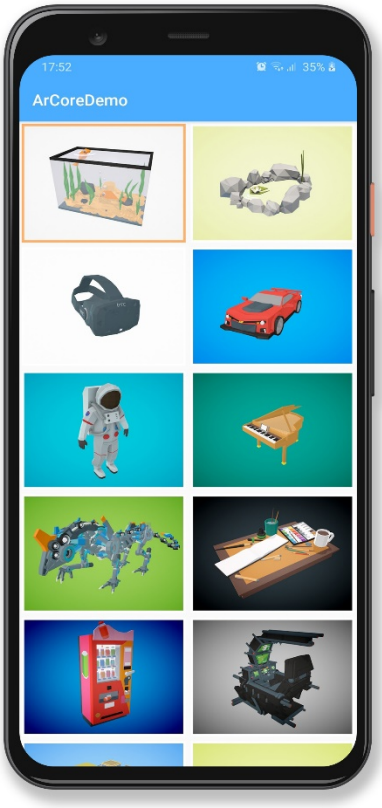
Programski kod 3.13. Dohvaćanje podataka iz glavne aktivnosti.

Jedini problem predstavlja format podataka. ItemAdapter drži listu Item objekata, dok PolyAPI vraća PolyResponse objekt. Nepotrebno je cijeli odgovor predati adapteru pa se zbog toga koristi klasa PolyParser (Programski kod 3.14.). Ona oblikuje i vraća format koji se lako može predati u ItemAdapter. Formatiranje se obavlja na način da se prolazi po cijeloj dobivenoj listi. Za svaki model

stvara se novi Item, unutar kojega se spremaju ID, poveznica na njegovu minijaturu i datoteka potrebna za 3D prikaz. Po završetku metode vraća se lista generiranih Item objekata.

```
public static List<Item> parseListAssets(PolyResponse responseBody) {
    List<Item> items = new ArrayList<>();
    List<Asset> assets = responseBody.getAssets();
    Asset helper;
    Format formatHelper;
    for(int i = 0; i < assets.size(); i++) {
        helper = assets.get(i);
        Item item = new Item(helper.getName());
        String url = helper.getThumbnail().getUrl();
        item.setThumbnail(url);
        List<Format> formats = helper.getFormats();
        for(int j = 0; j < formats.size(); j++) {
            formatHelper = formats.get(j);
            if(formatHelper.getFormatType().equals("GLTF2")){
                item.setModelUrl(formatHelper.getRoot().getUrl());
            }
        }
        items.add(item);
    }
    return items;
}
```

Programski kod 3.14. Stvaranje popisa prikladnog za ItemAdapter.



Slika 3.4. Prikaz glavne aktivnosti. [30]

3.4. Korisničko iskustvo

Aplikacija, za sada, korisniku ne omogućava prilagodbu pretraživanja, već samo odabir modela s automatskog prikaza. Ne postoji opcija za pretragu ili AR pregled. Korisnik također nije obaviješten ukoliko dođe do mrežnih ili bilo kojih drugih problema. Kako bi se uistinu omogućilo korištenje aplikacije, potrebno je ugraditi tipke za pokretanje dodatne funkcionalnosti. Unutar glavne aktivnosti dodane su dvije, ViewIn3D i SearchModels. Jedna služi za pokretanje AR iskustva, dok druga omogućuje ponavljanje API upita od strane korisnika. Kako bi se to omogućilo, potrebno je na tipku SearchModels postaviti onClickListener i unutar onClick metode implementirati željenu logiku. (Programski kod 3.15.)

```
private Button buttonSearch = buttonSearch
                              .setOnClickListener(this::onSearch);
private void onSearch(View view) {
    View search = this
            .getLayoutInflater()
            .inflate(R.layout.search_popup, (ViewGroup) view.getParent(), false);
    EditText editText = search.findViewById(R.id.keyword);
    AlertDialog alertDialog = new AlertDialog.Builder(this)
            .setTitle("Search 3D Models")
            .setView(search)
            .setPositiveButton("Search", (dialog, which) -> {
                String keyword = editText.getText().toString();
                snackBarHelper.showTimedMessage(
                        MainActivity.this,
                        getString(R.string.searchingModels));
                PolyAPI.callAPI(keyword, 40);
            })
            .setCancelable(true)
            .setNegativeButton("Cancel", (dialog, which) -> dialog.dismiss())
            .create();
    alertDialog.getWindow()
            .setSoftInputMode(
                    WindowManager.LayoutParams.SOFT_INPUT_STATE_VISIBLE);
    alertDialog.show();
}
```

Programski kod 3.15. Rukovanje pritiskom na tipku Search

Pritiskom tipke generira se novi AlertDialog [31]. Pomoću getlayoutinflater metode stvara se novi pogled. U njega se učitava .xml datoteka pod nazivom search_popup, unutar koje je definirano jedno EditText polje. Ono će se koristiti za unos ključnih riječi po kojima korisnik želi pretraživati. Nakon toga, novostvoreni pogled postavljamo kao zadani pomoću setView metode. AlertDialog također unutar sebe može postavljati tipke. Programski kod 3.15. opisuje dvije. Obje moraju imati onClick metode. Ovaj primjer postavlja onClick metode koristeći lambda izraze. Oni poboljšavaju

čitljivost i osiguravaju lakše razumijevanje koda. Programski kod 3.16. pokazuje primjer onClickListenera bez korištenja lambda izraze, dok primjer ispod njega pokazuje suprotno.

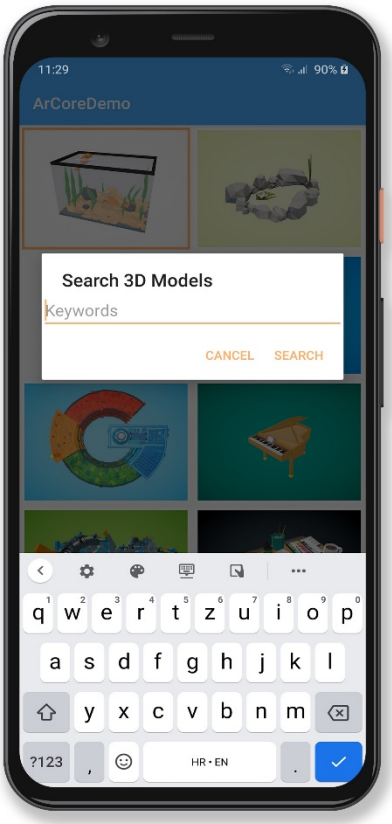
```
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //Some Logic here
    }
});
```

Programski kod 3.16. onClickListener metoda bez korištenja lambda izraza.

```
button.setOnClickListener(v -> { /* Some Logic here */ });
```

Programski kod 3.17. onClickListener uz pomoć lambda izraza.

Prvo se postavlja tipka pod imenom Search koristeći setPositiveButton metodu. Njenim pritiskom dohvaća se ključna riječ iz sadržaja EditText polja i upućuje upit na PolyAPI. Drugo, stvara se tipka Cancel koja omogućava prekid unosa bez pretraživanja. Koristeći metodu setSoftInputMode osigurava se pojavljivanje Android tipkovnice u trenutku pojavljivanja dijaloga za unos. Na samome kraju dijalog se prikazuje pozivajući na njemu show metodu.



Slika 3.5. Prikaz korisničkog sučelja za pretraživanje.

Iako je korisniku omogućeno pretraživanje i odabir željenog modela, još uvijek ne može pokrenuti AR iskustvo. Kako bi mu se pružila ta funkcionalnost, potrebno je dodati još jednu tipku. Kako bi se korisniku omogućilo što bolje iskustvo, potrebno je razmotriti problematiku kompatibilnosti. Osim ograničenja postavljenih stanjem Android fragmentacije, postoji ograničenje postavljeno od strane ARCore i Sceneform alata [32]. Podržani su samo uređaji posebno certificirani i kalibrirani od strane Google tima. S jedne strane ta činjenica je pozitivna jer osigurava konzistentnost iskustva, dok je s druge negativna jer drastično ograničava broj podržanih uređaja. Ukoliko je zamišljeno da aplikacija bude ograničena na tako uzak skup uređaja, nije potrebno vršiti dodatne provjere. Ipak, radi veće kompatibilnosti s uređajima koji ne podržavaju proširenu stvarnost, osiguran je alternativni prikaz 3D modela koji koristi nešto jednostavnije metode. Stoga je potrebno provjeriti podržava li uređaj ARCore. Klasa Utility provjerava dostupnost i frekvencijom od pet herca šalje upite sve dok je dostupnost i dalje nepoznata. (Programski kod 3.18.)

```
public final class Utility {
    private Utility() {}
    public static boolean ArCompatible(Activity activity) {
        ArCoreApk.Availability availability =
                ArCoreApk
                        .getInstance()
                        .checkAvailability(activity);
        if (availability.isTransient()) {
            new Handler().postDelayed(() -> ArCompatible(activity), 200);
        }
        return availability.isSupported();
    }
}
```

Programski kod 3.18. Provjera podržanosti ARCore alata.

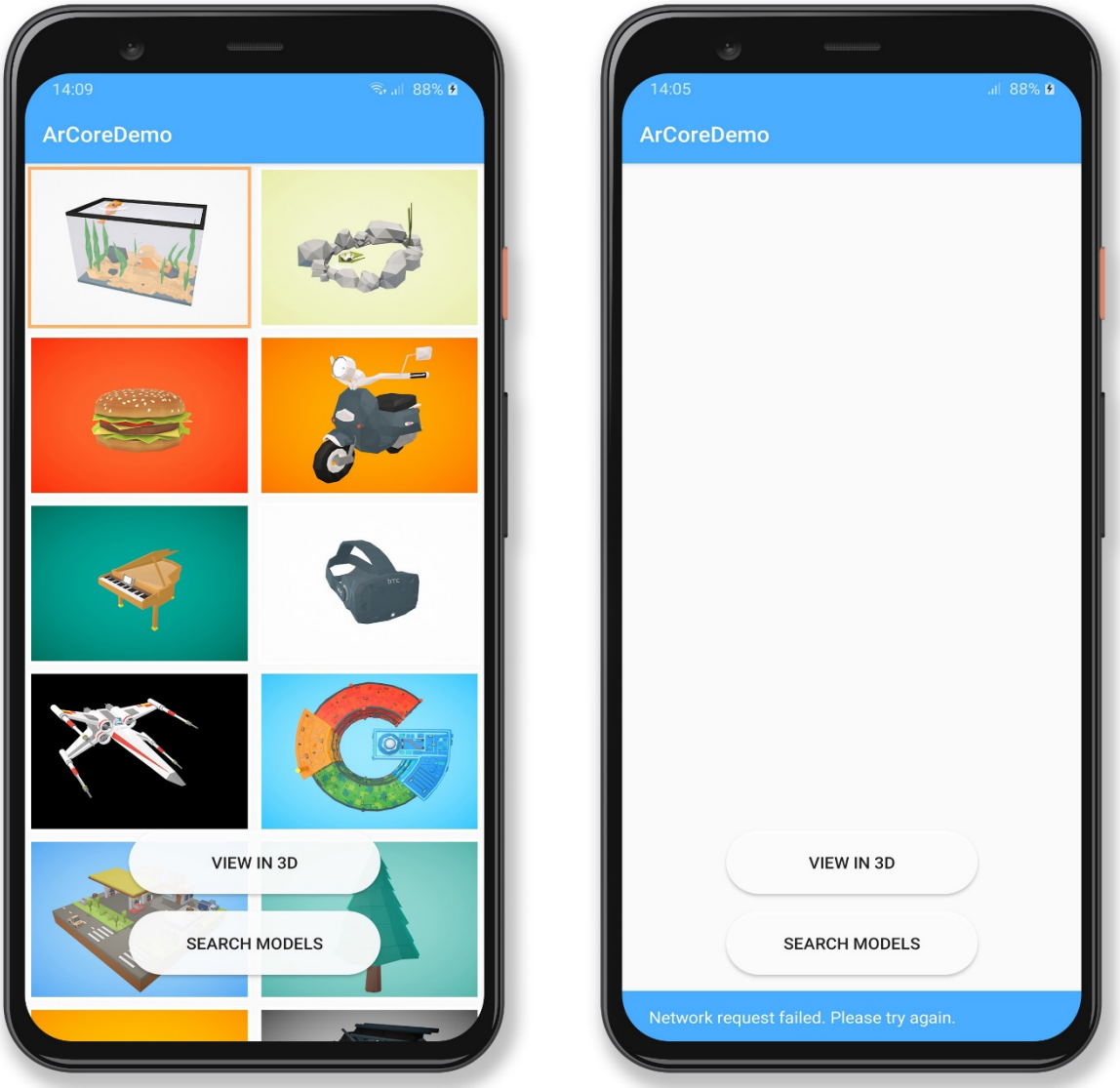
Ovisno o povratnoj informaciji klase Utility moguće je postaviti drugu tipku unutar glavne aktivnosti. Ukoliko uređaj podržava ARCore, otvara se nova aktivnost koja rukuje AR iskustvom, a u suprotnom podatci se šalju Google paketu za pretraživanje. On pomoću SceneViewer preglednika prikazuje model. SceneViewer je složeni preglednik koji omogućava dohvaćanje i prikaz podržanih 3D modela unutar Android aplikacije ili pomoću internetskog preglednika [33]. Moguće ga je podesiti da rukuje samo prikazom 3D modela s i bez korištenja proširene stvarnosti. Ukoliko AR integracija ne traži pretjerane modifikacije u usporedbi s mogućnostima SceneViewer preglednika, većini inženjera preporučuje se njegovo korištenje. U suprotnome potrebno je izraditi vlastiti preglednik i način crtanja grafike. U kontekstu ovoga rada, SceneViewer se ipak koristi samo za povećanje broja podržanih uređaja. Programski kod 3.19. prikazuje postavljanje logike prilikom pritiska tipke ViewIn3D.

```
private void setARListeners() {
    buttonAR.setOnClickListener(v -> {
        if (adapter != null) {
            selectedObject = adapter.getSelected().getModelUrl();
            Intent arIntent = new Intent(MainActivity.this, ArActivity.class);
            arIntent.putExtra("fileName", selectedObject);
            startActivity(arIntent);
        } else {
            snackBarHelper.showTimedMessage(this, getString(R.string.noModels));
        }
    });
}
private void setNonARListeners() {
    buttonAR.setOnClickListener(v -> {
        if (adapter != null) {
            selectedObject = adapter.getSelected().getModelUrl();
            Intent sceneViewerIntent = new Intent(Intent.ACTION_VIEW);
            sceneViewerIntent.setData(Uri.parse(selectedObject));
            sceneViewerIntent.setPackage("com.google.android.googlequicksearchbox");
            startActivity(sceneViewerIntent);
        } else {
            snackBarHelper.showTimedMessage(this, getString(R.string.noModels));
        }
    });
}
```

Programski kod 3.19. Postavljanje funkcionalnosti tipke ViewIn3D.

Prikazano u nekoliko zadnjih izvadaka koda moguće je primijetiti objekt pod nazivom snackBarHelper. Njega se koristi kako bi se korisniku prikazale bitne poruke tijekom životnog ciklusa cijele aplikacije. On enkapsulira logiku Android komponente Snackbar [34]. Koristeći njegove javno izložene metode mogu se bez pisanja velike količine koda, prikazati obavijesti o trenutnom stanju aplikacije ili dati upute za rukovanje elementima prikazanim na zaslonu. Izložene javne metode su showMessage, showTimedMessage, showDismissableMessage i showErrorMessage. Prva metoda trajno postavlja obavijest na zaslonu koju je jedino moguće ukloniti postavljanjem nove obavijesti. Druga postavlja poruku koja se sama uklanja nakon određenog vremenskog intervala. Treća postavlja trajnu poruku koju korisnik može ukloniti pritiskom na tipku. Četvrta pokazuje poruku pogreške i treba ju koristiti jedino ukoliko se aktivnost nikako ne može nastaviti. Metoda showErrorMessage nakon prikaza poruke ujedno i završava aktivnost na kojoj je pozvana.

Slika 3.6. prikazuje konačni izgled glavne aktivnosti i jednu od mogućih informativnih poruka. One uvelike olakšavaju korištenje sučelja. Obavještavaju u slučaju problema, a u AR aktivnosti služit će za edukaciju i pojašnjavanje interakcije s 3D objektima.



Slika 3.6. Konačni izgled glavne aktivnosti i prikaz obavijesti.

3.5. ARCore i Sceneform kompleti za razvoj aplikacija proširene stvarnosti

ARCore alat, s jedne strane, vrlo dobro rješava problem detekcije površina, praćenja objekata i lokalizacije korisnika, dok s druge strane uopće ne nudi metode kojima se 3D modeli mogu prikazati. Sva službena dokumentacija, koja bi trebala inženjerima biti primjer najjednostavnijeg prikaza modela, koristi isključivo OpenGL. Nakon višesatnog istraživanja i proučavanja zaključeno je da za potrebe završnog rada tj. izrade jednostavne aplikacije mora postojati lakši pristup [35].

Sceneform [36] se iskazao kao prihvatljivo rješenje za učitavanje i rukovanje modelima. Pozadinski koristi OpenGL, stoga su performanse zadovoljavajuće. Kao svoje sastavne dijelove nudi:

- API visoke razine apstrakcije pomoću kojega je moguće upravljati AR scenom.
- Apstrakciju Fillament sustava za iscrtavanje računalne grafike koristeći principe fizičkog modeliranja ponašanja svjetlosti.
- Učitavanje modela tijekom izvođenja programa.
- Android Studio dodatak za učitavanje i optimizaciju modela.

Treća točka popisa je izrazito važna. Kako se prije konstatiralo, učinkovitije je modele dohvaćati tijekom izvođenja programa. Dinamičko mijenjanje modela velika je prednost koju Sceneform pruža, ali podržava i učitavanje unaprijed pripremljenih modela koristeći Android Studio dodatak. Ovisno o primjeni, nekada je bolji pristup osigurati da aplikacija ne mora imati pristup internetu.

Ukoliko se pristupi statičkoj implementaciji modela, bitno je koristiti Sceneform veziju 1.15.0. Dodatak za Android Studio koji se koristi za učitavanje modela, nestabilan je u kombinaciji s novijim verzijama. Osim dodatka nestabilan je i sam Android Studio, ukoliko se koristi verzija novija od 3.5.3. Dinamičko učitavanje pomoću Sceneform 1.15.0 je i dalje stabilno na svim verzijama razvojnog okruženja. Nije poznato koliko dugo će se zadržati postojeće stanje, pošto službena podrška više ne postoji. Potrebno je napomenuti da sve novije inačice više neće biti održavane od strane Google tima. Sceneform 1.16.0 i kasnije varijante su službeno otvorenog izvornog koda [37].

Svaka aplikacija koja primjenjuje ARCore tehnologiju, može se postaviti na dva različita načina. Aplikaciju je moguće proglasiti kao *AR Required*, što značili da ju mogu instalirati samo

uređaji koji podržavaju ARCore. Drugi način je postavljanje aplikacije kao *AR Optional*. Takve aplikacije mogu instalirati svi uređaji s podržanim operacijskim sustavom. Neće svi korisnici biti u mogućnosti pristupiti svim značajkama aplikacije, ali neće biti niti potpuno zaboravljeni. Nužno je omogućiti pristup kameri i postaviti ovisnost o OpenGL verziji 3.0 koja je potrebna za Sceneform. Programski kod 3.20. prikazuje potrebne modifikacije unutar AndroidManifest.xml datoteke.

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature
    android:glEsVersion="0x00030000"
    android:required="true" />
<application
        .
        .
        .
    <meta-data
        android:name="com.google.ar.core"
        android:value="optional" />
</application>
```

Programski kod 3.20. Postavljanje AndroidManifest.xml datoteke.

Ukoliko je na uređaju podržan ARCore, potrebno je otvoriti novu aktivnost koja će upravljati elementima proširene stvarnosti. Službena dokumentacija navodi kako je najbolje koristiti SceneViewer, ali taj pristup je poprilično manjkav. Moguće je postaviti 3D model u kontekstu proširene stvarnosti koristeći SceneViewer. Omogućene su čak i geste za interakciju s objektom. (Dopušteno ga je povećati, smanjiti, rotirati itd.) Osim navedenoga, nema dodatnih funkcionalnosti; SceneViewer ne ostavlja prostor za fleksibilnost. (Npr. nije moguće dodati više od jednog 3D modela, niti je moguće na bilo koji način proširiti AR iskustvo.)

Za početak, potrebno je dodati ovisnosti o ARCore i Sceneform alatima unutar build.gradle datoteke (Programski kod 3.21). Sceneform koristi fragment za prikaz i konfiguraciju scene. Stvaranjem objekta ArFragment, automatski se vrše provjere potrebne za inicijalizaciju cijelog sustava. Ukoliko neki od uvjeta nije zadovoljen, ispisuje se potrebna poruka. (Npr. provjerava se instalirana verzija ARCore servisa.) Ukoliko servis nije instaliran, korisniku se prikazuje dijalog za instalaciju. Nakon što su uspješno obavljene sve provjere, stvara se objekt ArSceneView. Njegova površina služi za iscrtavanje kadrova dohvaćenih sa stražnje kamere. Također se automatski stvara Session objekt koji omogućava pristup podatcima položaja uređaja i podatcima dobivenim od kamere. Inicijalizira se PlaneRenderer objekt koji pomaže u vizualizaciji pronađenih površina. Sceneform automatski postavlja većinu objekata koje ARCore koristi.

```
implementation 'com.google.ar:core:1.15.0'
implementation 'com.google.ar.sceneform:assets:1.15.0'
implementation 'com.google.ar.sceneform.ux:sceneform-ux:1.15.0'
```

Programski kod 3.21. Ovisnosti potrebne za ARCore i Sceneform.

Sceneform vrši automatsku generaciju svih resursa potrebnih za pokretanje AR iskustva, jednako kao i SceneViewer. Ipak, vrlo se razlikuje po razini fleksibilnosti koju je moguće ostvariti. Sceneform omogućava inženjeru prilagodbu cijelog sustava i potpunu personalizaciju. Vrlo je jednostavno naslijediti klasu ArFragment i prepisati potrebne metode. Npr. automatska generacija objekata ne koristi Lighting Estimation API. Navedeni API koristi konvolucijsku neuralnu mrežu za analizu, procjenu intenziteta i smjera svjetlosti unutar scene [38]. Koristeći njene izlazne podatke moguće je pravilno postaviti svjetlinu i sjenu prikazanog objekta. Stoga je naslijeđena ArFragment klasa prepisana metoda koja omogućuje Light Estimation API. (Programski kod 3.22.)

```
public class CustomArFragment extends ArFragment {
    @Override
    protected Config getSessionConfiguration(Session session) {
        Config.LightEstimationMode lightEstimationMode =
                Config.LightEstimationMode.ENVIRONMENTAL_HDR;
        Config config = new Config(session);
        config.setLightEstimationMode(lightEstimationMode);
        return config;
    }
}
```

Programski kod 3.22. Stvaranje prilagođene ArFragment klase.

Zadatak AR aktivnosti je stvoriti i povezati ArFragment s pripadajućom .xml datotekom. Sva ostala logika delegirana je na SceneHelper klasu. Pomoću metode setObject klasi se predaje objekt koji treba prikazati. Postavlja se tipka za povrat u glavnu aktivnost, u kojoj je moguće odabrati drugi model za prikaz.

```
public class ArActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_ar);
        init();
    }
    private void init() {
        Button button = findViewById(R.id.btnBack);
        button.setOnClickListener(v -> finish()); Intent intent = getIntent();
        String selectedObject = intent.getStringExtra("fileName");
        CustomArFragment fragment = (CustomArFragment)getSupportFragmentManager()
                        .findFragmentById(R.id.ux_fragment);
        SceneHelper sceneHelper = new SceneHelper(fragment);
        sceneHelper.setObject(selectedObject); }}
```

Programski kod 3.23. Stvaranje fragmenta i delegacija odgovornosti na SceneHelper klasu.

Unutar konstruktora SceneHelper klase (Programski kod 3.24.) potrebno je iz predanog fragmenta dohvatiti reference na objekte koji se koriste za stvaranje AR pregleda. SceneView objekt obavlja integraciju s ARCore funkcionalnostima i prikazuje scenu. *Scene* objekt unutar sebe održava hijerarhijski prikaz prikazanih modela. U obliku stabla scena može imati niti jednog, jednog ili više nasljednika. Neovisno o sceni i svaki prikazani model ili ključna točka može imati niti jednog, jednog ili više nasljednika. Uobičajeno je da se modeli postavljaju kao djeca scene. *Camera* predstavlja virtualnu kameru koju je moguće pozicionirati bilo gdje u prostoru, ali u ovome kontekstu predstavlja fizičku kameru uređaja. Pomoću nje se prati pozicija korisnika u prostoru. ModelRenderable se koristi za iscrtavanje 3D modela nakon njegova dohvaćanja i stvaranja, dok numberOdArchorNodes prati broj trenutno prikazanih modela. SnackBarHelper se koristi za ispisivanje poruka korisniku. Na samome kraju poziva se setupFragment metoda kojom se postavljaju dodatne prilagodbe AR prikaza.

```
private SnackBarHelper snackBarHelper;
private CustomArFragment fragment;
private ArSceneView sceneView;
private PlaneRenderer planeRenderer;
private Scene scene;
private Camera camera;
private ModelRenderable modelRenderable;
private int numberOfAnchorNodes = 0;
public SceneHelper(CustomArFragment fragment) {
    this.fragment = fragment;
    this.sceneView = fragment.getArSceneView();
    planeRenderer = sceneView.getPlaneRenderer();
    this.scene = sceneView.getScene();
    this.camera = scene.getCamera();
    snackBarHelper = new SnackBarHelper();
    snackBarHelper
            .showMessage(
                    fragment.getActivity(),
                    fragment.getString(R.string.searchForSurface));
    setupFragment();
}
```

Programski kod 3.24. Stvaranja SceneHelper objekta.

Metoda setupFragment postavlja onUpdateListener na scenu (Programski kod 3.25.). Sceni se daje do znanja, da prilikom iscrtavanja svakog kadra kamere, pokrene onUpdate metodu. Specifično, provjerava nalazi li se korisnik svojom pozicijom unutar nekog od postavljenih 3D modela, a nakon toga na zaslonu ispisuje ključne informacije ovisno o stanju scene. Korisniku je potrebno demonstrirati da kameru blago pomakne naprijed, natrag, kako bi ARCore mogao početi prikupljati i uspoređivati prostorne podatke. Informacija o nužnosti „skeniranja“ dostupnih površina prikazuje se korištenjem jednostavne animacije potrebnog pokreta. Nakon pronalaska minimalno jedne površine,

animacija se uklanja i postavlja se nova poruka. Pronađene površine se vizualiziraju u obliku mreže bijelih točaka (Slika 3.7.).

Dodatno, na fragment se postavlja OnTapPlaneListener. Pomoću te metode osigurava se prepoznavanje i obrađivanje svakog korisnikovog dodira zaslona. Prilikom dodira pokreće se metoda onTapPlane, unutar koje se rukuje stvaranjem i postavljanjem 3D modela.

```
private void setupFragment() {
    scene.addOnUpdateListener(this::onUpdate);
    fragment.setOnTapArPlaneListener(this::onTapPlane);
}
private void onUpdate(FrameTime frameTime) {
    warnIfInsideObject(checkForCollision());
    showInstructions();
    renderDetectedPlane();
}
```

Programski kod 3.25. Prilagodba scene.

ARCore koristi podatke prikupljene od strane kamere za stvaranje najbolje procjene raspoloživog prostora. Ukoliko je otkrivena minimalno jedna horizontalna površina, korisnika se obavještava novom porukom (Programski kod 3.26.). Sada je moguće postaviti model.

Slika 3.7. Upućivanje korisnika i vizualizacija pronađene ravnine.

```
private void showInstructions() {
    if (snackBarHelper.getMessage().equals(fragment.getString(R.string.searchForSurface))) {
        for (Plane plane : sceneView.getSession().getAllTrackables(Plane.class)) {
            if (plane.getTrackingState() == TrackingState.TRACKING
                    &&
                    plane.getType().equals(Plane.Type.HORIZONTAL_UPWARD_FACING))
            {
                snackBarHelper
                        .showDismissibleMessage(
                                fragment.getActivity(),
                                fragment.getString(R.string.tapInstruction));
                break;
            }
        }
    }
}
```

Programski kod 3.26. Promjena poruke nakon otkrivanja površine.

Površinu koju je prepoznao ARCore dobro je vizualizirati korisniku. Tome služi renderDetectedPlane metoda. Ukoliko nije pozicioniran niti jedan objekt, poželjno je prikazati sav dostupan prostor, a nakon postavljanja, ukloniti vizualizaciju ravnine, kako bi se povećala realnost scene. Funkcionalnost je postignuta modifikacijom PlaneRenderer objekta (Programski kod 3.27.). Pristupom materijalu korištenom za vizualizaciju ravnine, moguće je promijeniti zadane postavke. Dovoljno je postaviti radijus prikazivanja, koji je originalno postavljen na svega nekoliko metara. Prilagođen radijus prikazivanja ravnine postavljen je na sto metara od srednje točke ravnine.

```
private void renderDetectedPlane() {
    if(!planeRenderer.isVisible()) return;
    else if(numberOfAnchorNodes == 0)
        planeRenderer
                .getMaterial()
                .thenAccept(
                        material -> {
                            material
                                .setFloat(
                                        PlaneRenderer.MATERIAL_SPOTLIGHT_RADIUS, 100f);
                        });
    else planeRenderer.setVisible(false);
}
```

Programski kod 3.27. Prilagodba radijusa prikazivanja otkrivene površine.

Zadnji kod koji se izvodi prilikom svakog iscrtavanja zaslona provjerava nalazi li se korisnik unutar nekog od već postavljenih objekata. Ovakve situacije dobro je izbjeći, ali, ukoliko ipak to nije moguće, potrebno je korisnika obavijestiti (Programski kod 3.29.). Originalno je zamišljeno uključiti vibracijski motor sve dok korisnik ne pomakne sebe ili objekt. Ispostavilo se da to nije najbolje rješenje iz nekoliko razloga. Prvo, korisnici su bili vidno uznemireni konstantnom vibracijom. Drugo, ponekad zbog vibracije uređaja ARCore izgubi prostorno shvaćanje i praćene objekte. Nepoželjni

rezultati su u oba slučaja. Stoga, umjesto vibracije, pušten je jednostavan ton u trajanju od trideset pet milisekundi frekvencije tisuću šesto megaherca. Osim toga, prikazana je obavijest korisniku pomoću snackBarHelper objekta. Metoda checkForCollision (Programski kod 3.28.) konstantno provjerava korisnikovu udaljenost od svih postavljenih 3D modela. Ray i HitTestResult objekti omogućuju provjeru korisnikove lokacije. *Ray* predstavlja polupravac u trodimenzionalnom prostoru, a HitTestResult omogućuje ispitivanje svojstva bilo koje linije. Prilikom svakog iscrtavanja stvara se novi polupravac, čije se ishodište nalazi u trenutnom položaju kamere uređaja. Njegov vektor smjera postavljen je korištenjem getForward metode na objektu kamere, koja vraća trodimenzionalni vektor. Nakon generacije polupravca testiraju se njegova svojstva u odnosu na scenu. Ukoliko on prolazi kroz bilo koji od postavljenih 3D modela, mjeri se udaljenosti od ishodišta do modela. Za slučajeve kada su svi navedeni uvjeti zadovoljeni i udaljenost je manja od jednog milimetra, metoda checkForCollision vraća *true* povratnu vrijednost.

```
private boolean checkForCollision() {
    Ray ray = new Ray(camera.getWorldPosition(), camera.getForward());
    HitTestResult hitTestResult = scene.hitTest(ray);
    return hitTestResult.getNode() != null && hitTestResult.getDistance() < 0.001;
}
```

Programski kod 3.28. Provjera sudara korisnika i objekta.

```
private void warnIfInsideObject(boolean collision) {
    if (collision) {
        if (!snackBarHelper.isShown()) {
            snackBarHelper
                    .showTimedMessage(
                            fragment.getActivity(),
                            fragment.getActivity().getString(R.string.insideModel));
            ToneGenerator toneGenerator = new ToneGenerator(AudioManager.STREAM_MUSIC, 100);
            toneGenerator.startTone(ToneGenerator.TONE_PROP_BEEP);
        }
    }
}
```

Programski kod 3.29. Slanje obavijesti korisniku.

Nakon svega navedenog obavljeni su preduvjeti za postavljanje i željeno prilagođavanje scene. Jedino je još potrebno korisniku omogućiti postavljanje modela u prostor i interakciju s njim. Zadatak koji, na prvi pogled, predstavlja najveći problem završnog rada, ali u stvarnosti postaje relativno jednostavan. Programski kod 3.25. ujedno postavlja i OnTapPlaneListener koji prilikom svakog dodira zaslona pokreće funkcionalnost definiranu unutar metode onTapPlane (Programski kod 3.30.).

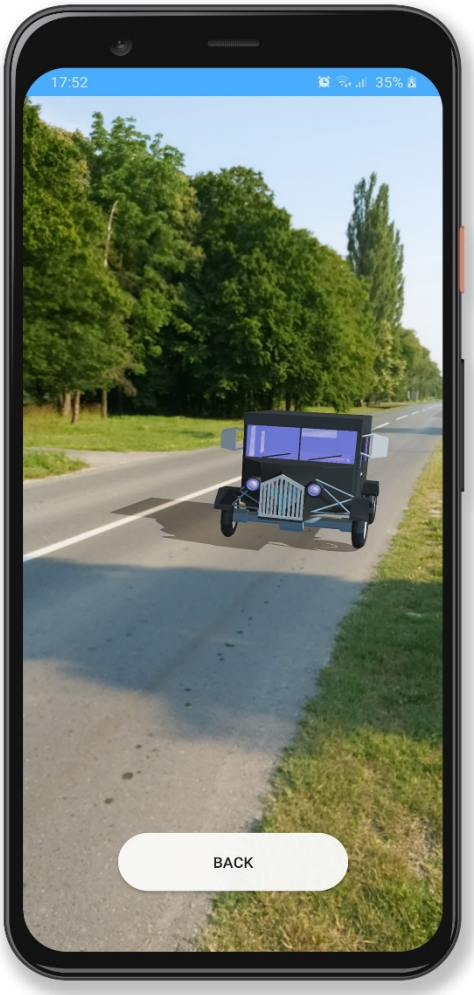
```
private void onTapPlane(HitResult hitResult, Plane plane, MotionEvent motionEvent) {
    if (plane.getTrackingState() == TrackingState.TRACKING
            && plane.getType() == Plane.Type.HORIZONTAL_UPWARD_FACING) {
        if (numberOfAnchorNodes == 0) {
            snackBarHelper.showTimedMessage(
                            fragment.getActivity(),
                            fragment.getString(R.string.nodeInstruction));
        }
        Anchor anchor = hitResult.createAnchor();
        AnchorNode anchorNode = new AnchorNode(anchor);
        anchorNode.setParent(scene);
        TransformableNode object = new TransformableNode(fragment.getTransformationSystem());
        object.setRenderable(modelRenderable);
        object.setParent(anchorNode);
        object.select();
        numberOfAnchorNodes++;
    }
    if (numberOfAnchorNodes > 5) {
        for (int i = 0; i < scene.getChildren().size(); i++) {
            if (scene.getChildren().get(i) instanceof AnchorNode) {
                scene.getChildren().get(i).setParent(null);
                numberOfAnchorNodes--; break;
            }
        }
    }
}
```

Programski kod 3.30. Metoda onTapPlane.

Primarno se vrši provjera stanja ravnine odabrane pritiskom zaslona. Ukoliko je ravnina horizontalna, njena normala usmjerena ka pozitivnom smjeru aplikate i ARCore može potvrditi trenutno praćenje njenih ključnih točaka, sigurno je postaviti 3D model. Odvija se sličan proračun kao i za detekciju udaljenosti korisnika od objekta. Razlikuje se jedino po tome što se računa udaljenost od sjecišta ravnine s polupravcem do točke ishodišta polupravca, tj. kamere uređaja. Njihovo sjecište označava se objektom Anchor koji ARCore kontinuirano nastavlja pratiti. Stvara se objekt specifičan za Sceneform, AnchorNode, koji se dodaje kao dijete sceni. On služi kao poveznica između lokacijskih i transformacijskih podataka. Nakon navedenog je poznata lokacija na kojoj će 3D model biti prikazan.

Kako bi se omogućila interakcija između korisnika i modela, potrebno je postaviti objekt koji rukuje transformacijama. TransformableNode rješava upravo taj problem. On ima već definirano rukovanje standardnim Android gestama poput zumiranja i rotiranja. Unutar njega postavlja se model u obliku modelRenderable reference. Kao roditelj postavlja mu se AnchorNode iz kojega se dohvaćaju svi relevantni podatci o lokaciji. Na samome kraju, ograničava se količina 3D objekata koje je moguće dodati u scenu, kako bi se spriječilo preopterećenje sustava i pregrijavanje uređaja.

Programski kod 3.23., kao svoju zadnju liniju, koristi jedinu javno izloženu metodu SceneHelper klase. Pomoću nje prosljeđuje se *String* koji u sebi sadrži URI objekta za prikaz. Pomoću RenderableSource objekta formatiraju se podatci o datoteci potrebnoj za prikaz 3D modela, kako bi bila prihvatljiva za daljnje korištenje. Jedini formati koji podržavaju ovakav način učitavanja su .glTF i .glb. Zbog toga se, prilikom dohvaćanja podataka API pozivom, filtriralo prema tipu datoteke (Programski kod 3.6.). Oni su manje optimizirani za prikaz naspram formata koje statički generira Sceneform dodatak za Android Studio, ali su fleksibilniji jer ne moraju biti unaprijed pripremljeni. Stvoreni objekt prosljeđuje se kao izvor podataka ModelRenderable.builder objektu koji preuzima potrebnu datoteku s interneta i ažurira modelRenderable referencu. Ona se prilikom dodira zaslona uređaja koristi za prikaz 3D modela u kontekstu proširene stvarnosti. Slika 3.8. prikazuje konačni rezultat nakon postavljanja modela.



Slika 3.8. Model postavljen u kontekstu proširene stvarnosti.

4. ZAKLJUČAK

Završnim radom su prikazane razlike između proširene i virtualne stvarnosti, navedene su prednosti i mane obje tehnologije. Na postojećim aplikacijama objašnjeno je par primjena proširene stvarnosti, navedeni su i kratko uspoređeni trenutno najpopularniji alati. Unutar implementacijskog dijela rada objašnjeno je postavljanje razvojnog okruženja, razlozi odabira minimalne podržane verzije operacijskog sustava i problemi koji nastaju kao posljedica. Prikazana je mrežna komunikacija koristeći API poziv i optimiziran pristup prikaza dobivenih rezultata. Kao opcija naveden je i statički način pripreme AR modela. Korisničko iskustvo je poboljšano olakšavanjem navigacije i prikazivanjem obavijesti. Detaljno su objašnjeni razlozi odabira pojedinih tehnologija i njihova primjena. Demonstrirano je kako uz pomoć ARCore i Sceneform alata prikazati 3D model u kontekstu proširene stvarnosti. Detaljno je razrađena implementacija korištenih verzija alata i upozoreno je na moguće probleme koji mogu nastati prilikom razvoja u budućnosti. Rezultati implementacijskog dijela su prikazani u obliku snimki zaslona aplikacije.

Završni rad mogao bi se nadograditi prilagodbom dubinskog razumijevanja prostora. Iscrtani modeli u trenutnoj verziji aplikacije, posjeduju ograničeno razumijevanje dubine. U slučaju kada bi se između korisnika i modela nalazile prepreke poput ograde, mreže, stupova itd., 3D modeli bili bi iscrtani preko njih. Očekivano ponašanje za stvarne objekte je da djelomično nestanu iza mogućih opstrukcija. Implementacijom Google Depth API tehnologije taj problem mogao bi se razriješiti. Poželjno bi bilo omogućiti korisniku dodavanje više objekata različitih vrsta unutar AR okruženja. Veličina modela dobrim dijelom ovisi o postavkama korištenim prilikom učitavanja u Poly servis, stoga je moguć nesrazmjer iscrtanih 3D objekata. Konstruktivna nadogradnja dodala bi algoritam prilagodbe veličine različitih modela kako bi mogli dijeliti isti prostor.

LITERATURA

[1] OssoVR, "The Leading, Virtual Reality Surgical Training & Assessment Platform," [Online]. Dostupno na: https://ossovr.com/. [25. lipnja 2020.].

[2] J. Psotka, "Immersive training systems: Virtual reality and education and training," *Instructional science,* vol. 23, no. 5-6, pp. 405-431, 1995.

[3] Valve Corporation, "A VR return to Half-Life," [Online]. Dostupno na: https://www.half-life.com/en/alyx/. [22. lipnja 2020.].

[4] Steam, "Valve Index VR Kit," [Online]. Dostupno na: https://store.steampowered.com/sub/354231/. [22. lipnja 2020.].

[5] Q. Sun, A. Patney, L. Y. Wei, O. Shapira, J. Lu, P. Asente, S. Zhu, M. McGuire, D. Luebke, and A. Kaufman, "Towards virtual reality infinite walking: dynamic saccadic redirection," *ACM Transactions on Graphics (TOG),* vol. 37, no. 4, pp. 1-13, 2018.

[6] Google LLC, "Share AR Experiences with Cloud Anchors," [Online]. Dostupno na: https://developers.google.com/ar/develop/java/cloud-anchors/overview-android. [23. lipnja 2020.].

[7] Google Developers, "Developing the First AR Experience for Google Maps (Google I/O'19)," Google LLC, [Online]. Dostupno na: https://www.youtube.com/watch?v=14wedZy90Tw. [23. lipnja 2020.].

[8] A. Benbihi, M. Geist, C. Pradalier, "ELF: Embedded Localisation of Features in pre-trained CNN," in *Proceedings of the IEEE International Conference on Computer Vision*, Seoul, South Korea, 2019.

[9] Google LLC, "Take off to your next destination with Google Maps," [Online]. Dostupno na: https://www.blog.google/products/maps/take-your-next-destination-google-maps/. [23. lipnja 2020.].

[10] Google LLC, "Augmented Reality," [Online]. Dostupno na: https://arvr.google.com/ar/. [23. lipnja 2020.].

[11] J. W. Brown, R. C. Malveau, H. W. McCormick III, and T. J. Mowbray, "Golden Hammer," in *Refactoring software, architectures, and projects in crisis*, John Wiley and Sons Inc, Canada, 1998, p. 63.

[12] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144,* 2016.

[13] Google Translate Help, "Translate what you find through your camera," [Online]. Dostupno na: https://support.google.com/translate/answer/6142483. [23. lipnja 2020.].

[14] X. Gu, "Google Translate's instant camera translation gets an upgrade," Google Translate, [Online]. Dostupno na: https://www.blog.google/products/translate/google-translates-instant-camera-translation-gets-upgrade/. [23. lipnja 2020.].

[15] Android developers, "Android Studio release notes 4.0," [Online]. Dostupno na: https://developer.android.com/studio/releases#4-0-0. [24. lipnja 2020.].

[16] Irina Galata, Joe Howard, Dick Lucas & Ellen Shapiro, Kotlin Apprentice Beginning programming with Kotlin, Razeware LLC, 2018.

[17] A. Ghuloum, M. B. Ari, "Fresher OS with Projects Treble and Mainline," Android Developers Blog, [Online]. Dostupno na: https://android-developers.googleblog.com/2019/05/fresher-os-with-projects-treble-and-mainline.html. [24. lipnja 2020.].

[18] Android Developers, "Documentation for app developers," Google LLC, [Online]. Dostupno na: https://developer.android.com/docs. [30. lipnja 2020.].

[19] Apple Inc., "App Store Support," [Online]. Dostupno na: https://developer.apple.com/support/app-store/. [24. lipnja 2020.].

[20] ALL3DP, "2020 Best 3D Scanners (Spring Update)," [Online]. Dostupno na: https://all3dp.com/1/best-3d-scanner-diy-handheld-app-software/. [24. lipnja 2020.].

[21] Google LLC, "Poly," [Online]. Dostupno na: https://poly.google.com/. [24. lipnja 2020.].

[22] Square Open Source, "Retrofit," [Online]. Dostupno na: https://square.github.io/retrofit/. [24. lipnja 2020.].

[23] Gradle Inc., "Gradle Build Tool," [Online]. Dostupno na: https://gradle.org/. [24. lipnja 2020.].

[24] D. Crockford, "Introducing JSON," [Online]. Dostupno na: https://www.json.org/json-en.htmlv. [24. lipnja 2020.].

[25] HexarA, "Json2Pojo," [Online]. Dostupno na: https://plugins.jetbrains.com/plugin/8533-json2pojo. [24. lipnja 2020.].

[26] E. Gamma, R. Helm, R. Johnson, J. Vlissides (Gang of Four), Design Patterns: Elements of Reusable Object-Oriented Software, Boston, Massachusetts: Addison-Wesley, 1994.

[27] Google LLC, "Google API Dashboard," [Online]. Dostupno na: https://console.developers.google.com/apis/dashboard. [25. lipnja 2020.].

[28] Android Developers, "Recyclerview," [Online]. Dostupno na: https://developer.android.com/jetpack/androidx/releases/recyclerview. [26. lipnja 2020.].

[29] Square Open Source, "Picasso," [Online]. Dostupno na: https://square.github.io/picasso/. [26. lipnja 2020.].

[30] nepoznat autor, "FreeMockUp," [Online]. Dostupno na: https://www.free-mockup.com/. [26. lipnja 2020.].

[31] Android Developers, "Documentation: AlertDialog.Builder," Google LLC, [Online]. Dostupno na: https://developer.android.com/reference/android/app/AlertDialog.Builder. [26. lipnja 2020.].

[32] ARCore Documentation, "ARCore supported devices," Google LLC, [Online]. Dostupno na: https://developers.google.com/ar/discover/supported-devices. [27. lipnja 2020.].

[33] ARCore Documentation, "Using Scene Viewer to display interactive 3D models in AR from an Android app or browser," Google LLC, [Online]. Dostupno na: https://developers.google.com/ar/develop/java/scene-viewer. [27. lipnja 2020.].

[34] Android Developer Documentation, "Snackbar," Google LLC, [Online]. Dostupno na: https://developer.android.com/reference/com/google/android/material/snackbar/Snackbar. [27. lipnja 2020.].

[35] E. Angel, D. Shreiner, "SIGGRAPH University : "An Introduction to OpenGL Programming"," SIGGRAPH Conference, [Online]. Dostupno na: https://www.youtube.com/watch?v=6-9XFm7XAT8. [27. lipnja 2020.].

[36] Sceneform Documentation, "Getting started with Sceneform," Google LLC, [Online]. Dostupno na: https://developers.google.com/sceneform/develop/getting-started. [27. lipnja 2020.].

[37] Google AR, "Sceneform Android SDK," Google LLC, [Online]. Dostupno na: https://github.com/google-ar/sceneform-android-sdk. [28. lipnja 2020.].

[38] C. LeGendre, W. C. Ma, G. Fyffe, J. Flynn, L. Charbonnel, J. Busch, and P. Debevec, "Deeplight: Learning illumination for unconstrained mobile mixed reality," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Long Beach, California, United States, 2019.

POPIS I OPIS UPOTRIJEBLJENIH KRATICA

- LLC (engl. *Limited liability company*) – d. o. o. (Društvo s ograničenom odgovornošću)
- iOS (engl. *iPhone operating system*) – Operacijski sustav korišten na iPhone uređajima.
- 3D – Trodimenzionalan, onaj koji posjeduje tri dimenzije.
- SDK (engl. *software development kit*) – Komplet alata i biblioteka za razvoj programa.
- IDE (engl. *Integrated development environment*) – Integrirano razvojno okruženje. Program koji služi za razvoj aplikacija.
- API (engl. *Application programming interface)* – Aplikacijsko programsko sučelje. Skup potprograma, gotovih funkcija i protokola koje programer može koristiti za kreiranje vlastitih programa.
- VR (engl. *Virtual reality*) – Virtualna stvarnost, računalno stvorena stvarnost.
- AR (engl. *Augmented reality*) – Stvarnost proširena računalno generiranim objektima postavljenim preko slike, najčešće generirane kamerom pametnog telefona.
- GPS (engl. *Global positioning system*) – Sustav koji se sastoji od skupa satelita, računala, odašiljača i prijemnika koji uspješno može odrediti lokaciju prijemnika na zemlji, računajući vremensku razliku u trajanju prijenosa signala između različitih satelita i prijemnika.
- 2D – Dvodimenzionalan, onaj koji posjeduje dvije dimenzije.
- HTTPS (engl. *Hypertext Transfer Protocol Secure*) – Mrežni protokol za prijenos podataka kojeg koriste izvorišno i odredišno računalo za uspostavu komunikacije.
- JSON (engl. *JavaScript Object Notation*) [24] – Jezik razmjene podataka, neovisan o platformi. Jednostavno ga je programski generirati i interpretirati. Jednako kako je jednostavno razumljiv računalima tako je razumljiv i ljudima. Pogledati prilog P.3.1. JSON podatci.
- Pojo (engl. *Plain old Java object*) – Objekt koji je ograničen jedino generalnim pravilima jezika. Sadrži javne metode postavljanja i dohvaćanja unutrašnjih članova. Ne smije deklarirati parametarski konstruktor, implementirati sučelja ili nasljeđivati druge klase.
- URI (engl. *Uniform resource identifier*) – Podatak koji jedinstveno određuje neki sadržaj.

SAŽETAK

Završni rad objašnjava razlike između proširene i virtualne stvarnosti. Navodi dva primjera upotrebe tehnologije proširene stvarnosti. Kratko spominje najpopularnije alate za izradu aplikacija koje se potpuno temelje na AR iskustvu ili ga samo dijelom implementiraju. Detaljno je objašnjena integracija mrežnog poziva i rukovanja mrežnim odgovorom u kontekstu Android aplikacije. Posebna pozornost posvećena je optimiziranom i održivom pristupu razvoja programske podrške. Navedeni su načini kojima se korisnik upućuje ka postizanju pozitivnog iskustva proširene stvarnosti. AR aplikacije moraju preuzeti dio edukacijske odgovornosti korisnika, pošto su intrinzično nove i nesvakidašnje. Navedeni su razlozi odabira ARCore i Sceneform alata. Na primjeru, korak po korak, prikazan je postupak razvoja AR iskustva i dinamičkog dohvaćanja i učitavanja 3D modela tijekom izvođenja programa. Konačno, opisani su problemi koji su nastali prilikom razvoja i način njihovog rješavanja.

Ključne riječi: Android, ARCore, AR model, Proširena stvarnost, Sceneform

ABSTRACT

Application and implementation of augmented reality technology on mobile platforms

The thesis briefly explains the differences between augmented and virtual reality. It lists a couple of usages of augmented reality. It mentions, briefly, the most popular SDKs and frameworks for AR application development without getting specific about the intrinsic application characteristics. API network calls and subsequent responses, in the context of an Android application, are thoroughly explained. A special attention is paid to optimization and sustainable software development practices throughout the project. The thesis lists the best design practices for ensuring a positive user experience. AR applications must assume a share of educational responsibility when it comes to a user's interaction. Additionally, the reasons for choosing ARCore and Sceneform SDKs are elaborated on. The implementation of an AR experience is demonstrated step-wise by using a practical example which consists of dynamical runtime 3D asset loading. Finally, problems that arose during development and their following resolutions are delineated.

Keywords: Android, ARCore, AR model, Augmented reality, Sceneform

ŽIVOTOPIS

Stjepan Mrganić, rođen je 25. studenog 1996. u Našicama. Dobitnik je nagrade Gradskog društva Crvenog križa za esej o toleranciji i nagrade Grada Zagreba „Luka Ritz – nasilje nije hrabrost“. Godine 2017. upisuje preddiplomski sveučilišni studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek pri Sveučilištu Josipa Jurja Strossmayera u Osijeku. Tijekom svog obrazovanja aktivan je član sveučilišnog IEEE studentskog ogranka u sklopu kojega održava besplatne grupne instrukcije programiranja studentima prve godine. Za vrijeme druge i treće godine studija obavlja poslove demonstratora iz kolegija Programiranje 1 i Programiranje 2.

Stjepan Mrganić

PRILOZI

P.3.0. Izvorni kod aplikacije

Dostupan na: https://github.com/smrganic/ArCoreDemo

P.3.1. JSON podatci


```
{
  "assets": [
    {
      "name": "assets/eke7qcu_FR2",
      "displayName": "Cheeseburger",
      "authorName": "Poly by Google",
      "description": "#hamburger #burger",
      "createTime": "2017-09-25T20:19:19.496101Z",
      "updateTime": "2020-06-24T10:12:52.002381Z",
      "formats": [
        {
          "root": {
            "relativePath": "Hamburger.obj",
            "url":
"https://poly.googleapis.com/downloads/fp/1592993572002381/eke7qcu_FR2/50NqG3OK3rn/Ham
burger.obj",
            "contentType": "text/plain"
          },
          "resources": [
            {
              "relativePath": "Hamburger.mtl",
              "url":
"https://poly.googleapis.com/downloads/fp/1592993572002381/eke7qcu_FR2/50NqG3OK3rn/Ham
burger.mtl",
              "contentType": "text/plain"
            },
            {
              "relativePath": "Hamburger_BaseColor.png",
              "url":
"https://poly.googleapis.com/downloads/fp/1592993572002381/eke7qcu_FR2/50NqG3OK3rn/Ham
burger_BaseColor.png",
              "contentType": "image/png"
            }
          ],
          "formatComplexity": {
            "triangleCount": "1095"
          },
          "formatType": "OBJ"
        },
```

```
        {
          "root": {
            "relativePath": "Hamburger.gltf",
            "url":
"https://poly.googleapis.com/downloads/fp/1592993572002381/eke7qcu_FR2/5SJ0FdoYxXK/Ham
burger.gltf",
            "contentType": "model/gltf+json"
          },
          "resources": [
            {
              "relativePath": "Hamburger.bin",
              "url":
"https://poly.googleapis.com/downloads/fp/1592993572002381/eke7qcu_FR2/5SJ0FdoYxXK/Ham
burger.bin",
              "contentType": "application/octet-stream"
            },
            {
              "relativePath": "Hamburger_BaseColor.png",
              "url":
"https://poly.googleapis.com/downloads/fp/1592993572002381/eke7qcu_FR2/5SJ0FdoYxXK/Ham
burger_BaseColor.png",
              "contentType": "image/png"
            }
          ],
          "formatComplexity": {
            "triangleCount": "1095"
          },
          "formatType": "GLTF2"
        }
      ],
      "thumbnail": {
        "relativePath": "thumbnail.png",
        "url": "https://lh3.googleusercontent.com/F1cb2DN34c8N38_h-
JbRbsg87V6btNJVupN5e2U5SHGgAsopnwCp8Fx5s8Mz8aE",
        "contentType": "image/png"
      },
      "license": "CREATIVE_COMMONS_BY",
      "visibility": "PUBLIC",
      "isCurated": true,
      "presentationParams": {
        "orientingRotation": {
          "w": 1
        },
        "colorSpace": "LINEAR",
        "backgroundColor": "#ff5722"
      }
    }
  ],
  "nextPageToken": "CiH6W___PCP1-
b6MjJqLjO3w9fLV9JqUmsiOnIqgua3N__4QASEDWc_7WxOaqjGhFGwarcBVGzncwwAApAAAAFAAWgsJag5eI0r
wJFIQA2CD3-CRAg==",
  "totalSize": 43
}
```