SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD

Kinematički model u okruženju proširene stvarnosti na ugrađenom sustavu

Petar Dučić

Zagreb, lipanj 2011.

Sadržaj

UVOD

Mobiteli su u današnje vrijeme postali svakodnevica. Ubrzanim razvojem tehnologije, ti isti mobiteli su postali prava prijenosna računala koja ostvaruju procesorsku snagu i preko 1GHz.

S druge strane imamo računalnu grafiku kao jedan od imperativa u prikazu podataka. Gotovo svaki popularniji mobilni uređaj omogućuje trodimenzionalni prikaz virtualnih modela. Također, kamera na mobitelima postala je nezaobilazni dodatak.

Navedene pojave u globalnim kretanjima na tržištu mobilnih uređaja i popratne tehnologije dovele su do omogućavanja proširene stvarnosti na ugrađenim sustavima poput mobitela.

Proširena stvarnost prisutna je na računalima od 90-ih godina 20. stoljeća. Koliko god to atraktivna tehnologija bila, nikad nije doživjela široku komercijalnu upotrebu upravo zbog robusnosti računala. Međutim, prethodno spomenutim procvatom mobilnih uređaja, te razvojem mobilnih platformi, omogućen je razvoj aplikacija baziranih na proširenoj stvarnosti programerima diljem svijeta. Kroz proteklih godinu dana razvijeno je par programskih radnih okvira (eng. framework*)* – kvalitetnih podloga za razvoj navedenih aplikacija. Također, prisutne su novčane nagrade za inovativne aplikacije.

Cilj ovog diplomskog rada je upravo proučiti spomenute programske okvire, te na osnovu primjera omogućiti razvoj sličnih aplikacija. Proučavat će se sama teorijska podloga proširene stvarnosti, mogućnosti definiranja okoline bez upotrebe markera, Android mobilna platforma, mogućnosti razvoja aplikacija temeljenih na proširenoj stvarnosti, dostupni *framevorci,* te sama implementacija jednostavnih primjera.

1. PROŠIRENA STVARNOST

1.1. DEFINICIJA

U osnovi definicije ovog frazema, proširena stvarnost (eng. Augmented reality – AR) predstavlja područje na granici stvarnog i virtualnog. Proširena stvarnost postavlja virtualne objekte u stvarni svijet.

Slika 1 – Proširena stvarnost

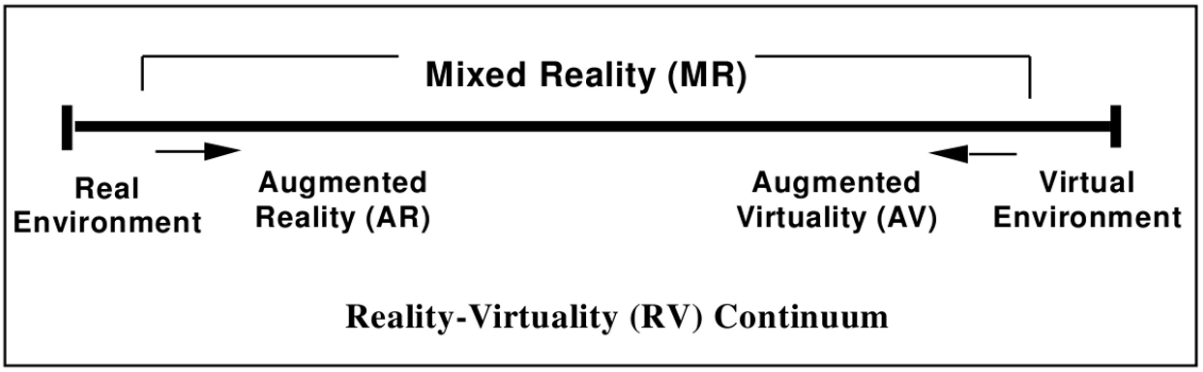
Službene i jedinstvene definicije proširene stvarnosti nema. Međutim, sustav koji predstavlja proširenu stvarnost, mora sadržavati sljedeće karakteristike:

1. Proširena stvarnost kombinira realno i virtualno
2. Omogućava interaktivni prikaz u realnom vremenu
3. Trodimenzionalnost

Prva karakteristika definira da će sustav kao rezultat prikazivati i dijelove virtualnog i dijelove stvarnog svijeta učitanog kroz kameru.

Drugom se karakteristikom proširena stvarnost ograničava na realno vrijeme. Ovom se točkom definicije iz proširene stvarnosti isključuju svi filmovi koji naizgled kombiniraju stvarno i virtualno, zbog nedostatka interaktivnosti.

Trećom se karakteristikom ograničavamo samo na sustave koji „osjećaju „ dubinu, te u odnosu prema njoj iscrtavaju virtualne objekte. Tako, primjerice prilikom fotografiranja, na ekranu nam se mogu pojaviti razne postavke prikazane virtualnim objektima - gumbima. Tu dolazi do miješanja stvarnosti i virtualnog okruženja u realnom vremenu. Međutim, prema ovoj postavci, takav prikaz ne spada u proširenu stvarnost. Dnevne vijesti su također dobar primjer sustava koji ne odgovara ovom svojstvu proširene stvarnosti.



Slika 2 – Kontinuum stvarnosti i virtualnosti

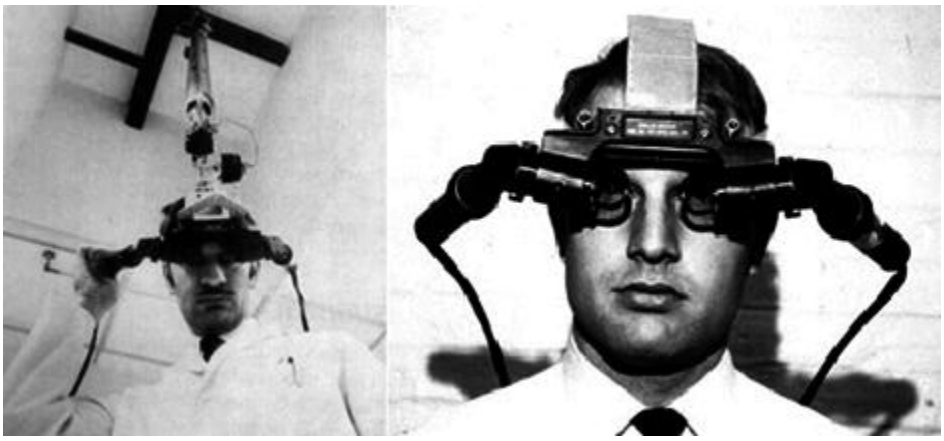
Slika 2 pokazuje kontinuum odnosa stvarnog i virtualnog okruženja predstavljena u [2]. Na jednoj strani ekstrema prikazano je stvarno okruženje, dok je na potpuno drugoj strani virtualna stvarnost. Sve unutar tih dviju stvarnosti pripada takozvanoj „miješanoj stvarnosti“. Prema ovoj podjeli, razlikujemo i „proširenu virtualnost“, gdje se stvarni objekti miješaju s virtualnim okruženjem. Primjer toga bi mogla biti virtualna soba u koju su dodane stvarne osobe.

Također, valja napomenuti da se proširena stvarnost može odnositi na sva osjetila, a ne samo na vid. Tako, na primjer, proširenu stvarnost možemo obogatiti zvukom. Slušalice bi mogle dodavati generirani zvuk, dok bi vanjski mikrofon prepoznavao zvukove iz okoline. Takav bi sustav omogućavao maskiranje i prekrivanje odabranih zvukova. Nadalje, s haptičkim uređajima koji pružaju povratne informacije možemo „proširivati“ stvarne sile iz okoline. Dakako, ovi

sustavi su komplicirani, ali ne i neizvedivi. Međutim, u ovom ćemo se radu baviti vizualnim sustavima proširene stvarnosti.

1.2. PROŠIRENA STVARNOST KROZ VRIJEME

Povijesno gledano, tragovi proširene stvarnosti sežu do davne 1897., kada se psiholog George Malcolm Stratton u jednom od svojih radova referencirao na „obrnute naočale“ (eng. upside-down glasses). Te su naočale čovjeku pružale obrnuti prikaz stvarnosti.



Slika 3 - Ivan Sutherland - HMD

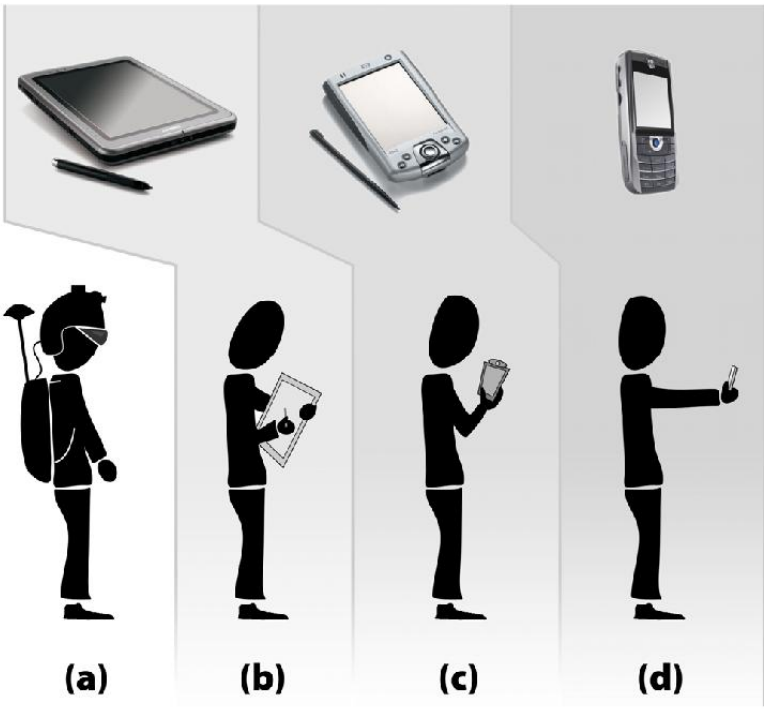
Iako se ovaj pristup čini prilično primitivnim, te uopće ne dolazi do „proširenja“ stvarnosti, uočljiva je sličnost sa sustavom Ivana Sutherlanda iz 1968. Sutherland je sa skupinom kolega dizajnirao HMD (eng. head mounted display) kroz koji je korisnik mogao vidjeti računalno generirane slike miješane sa stvarnim objektima (slika 3).

Kasnije, njegov je rad razvijala američka vojska kroz unaprjeđivanje prikaza vojnog pilota određenim informacijama. Njihov je rad u 90-tim godinama javno objavljen, te je postao osnova raznih civilnih razvojnih projekata.

Kakogod, termin „Proširene stvarnosti“ je prvi puta upotrjebljen u ljeto 1990. od strane Boeingovog razvojnog tima. Njihov je zadatak bio naći alternativu skupim dijagramima i markirnim uređajima korištenim za vođenje radnika kroz tvornicu

zaduženu za kabliranje. Rezultat istraživanja bio je uređaj koji se nosio na glavi, dizajniran da korisniku ispisuje instrukcije o kablovima aviona, te samim time olakšava i ubrzava rad u tvornici.

Pred kraj 90-ih, proširena je stvarnost postala prilično istraživana tema. Dr. Ronald Azuma je svojim radom „Pregled Proširene Stvarnosti“ (eng. „Survey of Augmented Reality“) 1997 postavio temelje ove grane znanosti.



Slika 4 - Sustavi korišteni za AR aplikacije (a): PC + HMD (b): Tablet PC (c): PDA (d): Smartphone

U narednim godinama, proširena je stvarnost postala sveprisutna u računalnom svijetu. Ostvarena su brojna tehnička ostvarenja na raznim sustavima. Pretežito je u svim takvim sustavima sastavna komponenta bilo zasebno računalo, te eksterne ulazne i izlazne jedinice. Međutim, u današnje doba, sve više sustava sadržavaju sve spomenute komponente u vlastitom integriranom sustavu. Tako smo od „robota“ kojeg smo morali nositi na sebi došli do pametnih mobitela koji se uz sve navedene komponente mogu pohvaliti i s preko 1 GHz procesne snage. Prema tome, tema proširene stvarnosti dobiva nove interese od strane sve brojnijih razvojnih timova specijaliziranih za mobilne aplikacije. Navedeno je područje uža tema ovog rada.

1.3. PRIMJENE PROŠIRENE STVARNOSTI

Proširena stvarnost broji razne praktične primjene. Od znanstvenih, medicinskih, onih ekonomske prirode, pa sve do područja zabave. U sljedećem će poglavlju biti navedene neke od praktičnih primjena proširene stvarnosti.

1.3.1. Dizajn interijera

Jedna od primjena proširene stvarnosti je dizajn namještaja. Ovakvim je sustavima omogućeno isprobavanje kako bi namještaj odgovarao u sobu prije samog kupovanja namještaja.

Slika 5 - Virtualni stol "ugrađen" u sobu

Također, moguća je primjena pri raspoređivanju već postojećeg namještaja. Umjesto stvarnog premještanja namještaja, dovoljno je premjestiti papir (marker) na željenu poziciju. Primjer ovakve primjene možemo vidjeti na slici 5.

1.3.2. Informacije o proizvodima

Širok je spektar načina informiranja kupaca o proizvodima putem proširene stvarnosti. Primjerice, Lego grupacija je započela s opremanjem svojih prodavaonica posebnim terminalima proširene stvarnosti. Takvi terminali omogućuju korisniku da vidi sadržaj kutije. Za prikaz takve proširene stvarnosti potrebno je samo prikazati kutiju kameri terminala. Terminal dalje na osnovu

markera prikazanog na kutiji analizira sadržaj kutije, te prikazuje trodimenzionalnu sliku na ekranu. Slika 6.

Slika 6 - Proširena stvarnost u pomoći korisnicima

1.3.3. Medicina

Proširena je stvarnost često korištena metoda u području medicine. Moguće je prikupiti podatke o pacijentu u stvarnom vremenu pomoću senzora, npr MRI(eng. magnetic resonance imaging), CT (eng. computed tomoghaphy), ili ultrazvuka. Takvi se podaci nadalje mogu prikazivati (eng. rendering) i kombinirati sa stvarnom slikom pacijenta. Rezultat ovakvog procesa davao bi doktoru pogled u tijelo pacijenta, kao na slici 7. Ovakav sustav može uvelike smanjiti potrebne intervencije.

Slika 7 - Proširena stvarnost u medicine

Također, proširena stvarnost bi u doglednoj budućnosti mogla uvelike pomoći u samoj kirurgiji. Međutim, ovakva primjena zahtjeva ogromne sigurnosne standarde i veliku preciznost.

1.3.4. Industrija

Mnoge vodeće tvrtke za razvoj automobila koriste složene metode planiranja i dizajniranja koje u sebi sadrže segmente proširene stvarnosti. U [5] se navodi primjena proširene stvarnosti u pogonima Volkswagena.

Također, neke firme omogućavaju svojim korisnicima pomoć pri samostalnom popravku ili održavanju proizvoda na osnovu proširene stvarnosti.

1.3.5. Zabava

Proširena stvarnost ipak svoju najširu primjenu nalazi u zabavi. Ostvarene su razne igre poput tenisa gdje se reketom upravlja pomoću markera, itd. Međutim, stvarno sve većom podrškom programera širom svijeta, i u ovom se sektoru nailazi na aplikacije koje uz zabavu nude i korisne informacije. Primjer takve aplikacije je Layar koji na osnovu GPS-a i orijentacije pametnog telefona iscrtava informacije o području gledanom kroz kameru. Primjer jednog takvog prikaza vidi

se na slici 8. S obzirom da se podaci svakodnevno ažuriraju, govorimo o jednoj od brže rastućih aplikacija za pametne telefone.

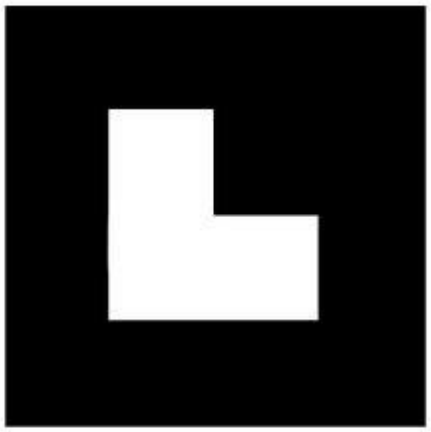
Slika 8 - Layar

1.4. SEGMENTI SUSTAVA PROŠIRENE STVARNOSTI

Tri su osnovna problema proširene stvarnosti, te se prema tome cijeli proces može segmentirati na:

- **Miješanje slike** (eng. mixing) je kombiniranje stvarne i virtualne slike. Razlikujemo optičko, video i projekciono miješanje. Optičko miješanje koristi polu prozirna ogledala – optičke miješalice ispred korisnika, na koju se projicira slika s monitora i kombinira sa stvarnim svijetom. Kod video miješanja se ne koristi optička miješalica, već se stvarni svijet snima, te se s virtualnim miješa u računalu. Tako proizvedena proširena stvarnost se projicira na ekran sustava. Također, razlikujemo još i projekciono miješanje koje se zasniva na prikazu na poseban zaslon ili projektor.
- **Poravnavanje** (eng. registration) je postupak pri kojem se virtualni objekti poravnavaju sa stvarnim objektima u prostoru. Konstruira se virtualna

scena s koordinatnim sustavom koji odgovara koordinatnom sustavu stvarnog svijeta. Time je omogućeno prebacivanje stvarnih objekata u virtualnu scenu, te samim time i postavljanje virtualnih objekata u odnosu na njih.



Slika 9 - Marker za optičko slijeđenje

- **Slijeđenje** (eng. tracking) predstavlja skup tehnika kojima se doznaje pozicija i promjena pozicije elemenata scene. Ovaj je postupak neophodan za prije spomenuti postupak poravnavanja. Postoje različite izvedbe ovog postupka: magnetsko, ultrazvučno, mehaničko, gps, te optičko praćenje koje nas najviše zanima. Takvo praćenje radi na principu detektiranja posebnih markera u slici koje računalo prima iz stvarnog svijeta. Primjer takvog markera možemo vidjeti na slici 9. Pomoću dobivene pozicije markera, računalo dobiva podatke o sceni. O jednoj ovakvoj implementaciji govorit će se u 4. poglavlju., Međutim, kroz zadnje godine radi se na prepoznavanju površina i prostora bez markera. Takav je proces računalno i algoritamski prilično zahtjevniji. O toj problematici i jednoj takvoj implementaciji govorit će se u 2. poglavlju ovog rada.

2. PRAĆENJE BEZ MARKERA

Razvoj prvih sustava za praćenje bez markera (eng. markerless visual tracking) započeo je istraživanjem C. Harrisa 1992. godine na MIT-u [7]. Njegov sustav, zvan RAPiD (eng. Real time attitude and position determination), bio je jedan od ranijih sustava za praćenje bez markera u tri dimenzije u stvarnom vremenu. Iako sa uvelike slabijim računalnim performansama, sustav je ostvario priličan uspjeh. Mnogi sustavi koji su uslijedili dijele iste principe rada kao i Harrisov sustav. Također, kratki opis načela rada ovog sustava poslužit će kao dobar uvod u tehnike praćenja bez markera.

U RAPiD-u sustav prati poziciju kamere u odnosu na 3D model. Pozicija je opisana sa šest parametara, po tri za translaciju, odnosno rotaciju. Za svaku sličicu videa (eng. frame), pozicija se ažurira: prvo po pretpostavkama na osnovu promjene položaja sustava, a zatim i na osnovu mjerenja iz video ulaza. Navedena mjerenja ostvaruju se na osnovu prepoznavanja rubova modela i mjerenja razlika između predviđenog i stvarnog položaja ruba. Generalno, razlike (greške), nikad ne mogu biti potpuno izolirane, međutim, mogu biti svedene na minimum.

Sustavi praćenja bez markera pretežito se oslanjaju na pronalaženje „prirodnih“ markera, onih detalja koji su mogu naći u svakoj realnoj sceni. Nadalje, nailazimo na problem oko otkrivanja dubinskog položaja. Neki sustavi su korištenjem dviju kamera prilično olakšali ovaj proces. Međutim, u nastavku će biti objašnjen rad jednog sustava koji koristi samo jednu kameru.

2.1. PTAM – PARALLEL TRACKING AND MAPPING

Ovaj se sustav koristi metodama za određivanje pozicije kamere u nepoznatoj sceni. Bazira se na tehnici SLAM (eng. simultaneous localization and mapping) koja se pretežito koristi u robotici jednake svrhe. PTAM sustav se bazira na podijeli praćenja i preslikavanja (mapiranja) u dva neovisna zadatka koje

omogućava razne metode optimizacije. Jedan proces se bavi praćenjem uređaja koje obavlja nepravilne pokrete, dok se u drugom generira trodimenzionalni koordinatni sustav (mapu) sa svim detaljima pronađenim u prethodnoj slici. U kasnijim poglavljima govori se o implementaciji ovog sustava, te se primjer jedne takve mape može vidjeti na slici 16.

Zbog česte nejasnoće elemenata na sceni, dolazi do razvoja tehnika pod skupnim nazivom „proširenо praćenje“ (eng. extensible tracking), u kojem sustav pokušava dodati prethodno nepoznate elemente scene u inicijalni trodimenzionalni koordinatni sustav. Budući da se ovaj sustav bazira na određivanju pozicije kamere u nepoznatoj okolini, ovakvo „proširenо praćenje“ počinje praktički iz ničega. Na kasnije opisanoj implementaciji, primjer takvog naknadnog praćenja vizualno se može primijetiti na slici 15.

2.1.1. Karakteristike metode u kontekstu SLAM tehnike

- Praćenje i preslikavanje su odvojeni, te se pokreću u odvojenim dretvama
- Preslikavanje scene u koordinatni sustav (mapiranje - postavljanje scene u koordinatni sustav) je zasnovano na „ključnim slikama“ (eng. keyframes)
- Koordinatni sustav je inicijaliziran iz para ključnih točaka (5-Point Algorithm)
- Novi detalji (točke) se inicijaliziraju na osnovu epipolarne geometrije
- Mapira se velik broj točaka – reda veličine i do 10^3

Ova se metoda pretežito temelji na traženju planarnih struktura – ravnih ploha. Također, implementacija ovog sustava omogućuje prikaz virtualnih objekata na jednoj od takvih ploha.

2.1.2. Izgradnja mape – koordinatnog sustava

Koordinatni sustav – u daljem tekstu „mapa“, sastoji se od M točaka lociranih u koordinatnom sustavu W. Svaka točka predstavlja lokalnu planarnu teksturu u svijetu – sceni koju promatramo. Svaka točka u mapi ima koordinate:

$$p_j = (x_j\ y_j\ z_j)^T$$

Također, svaka točka sadrži i normalu n_j i referencu na vezani detalj iz scene.

Nadalje, mapa sadrži i N ključnih slika (eng. keyframes). To su slike snimljene kamerom u različitim vremenima. Svaka ključna slika sadrži odgovarajući koordinatni sustav s kamerom pozicioniranom u centru. Svaki koordinatni sustav također sprema piramidu s četiri stupnja sivih bpp slika. Na slici 10 se nalazi primjer piramidalnog prikaza slike. Piramidalni prikaz slike sadrži slojeve slike u različitim rezolucijama. Tako je u sustavu PTAM na nultoj razini spremljena cijela slika rezolucije 640x480, dok su na trećem stupnju spremljena ista slika degradirane rezolucije na 80x60 piksela.

Slika 10 - Piramidalni prikaz slike

2.1.3. Praćenje

Prilikom praćenja, uz pretpostavku da je koordinatni sustav već izgrađen, sustav provodi slijedeću proceduru pri svakom okviru (*frameu)* ulaznog toka podataka:

1. Nova slika (eng. frame) se preuzima uz pomoć kamere. Izrađuje se piramidalni prikaz s četiri stupnja. Određuju se rubovi na osnovu algoritma FAST-10 za svaki stupanj piramide.
2. Točke koordinatnog sustava se projiciraju na sliku
3. Cca 50 najgrubljih detalja - točaka se traži na slici

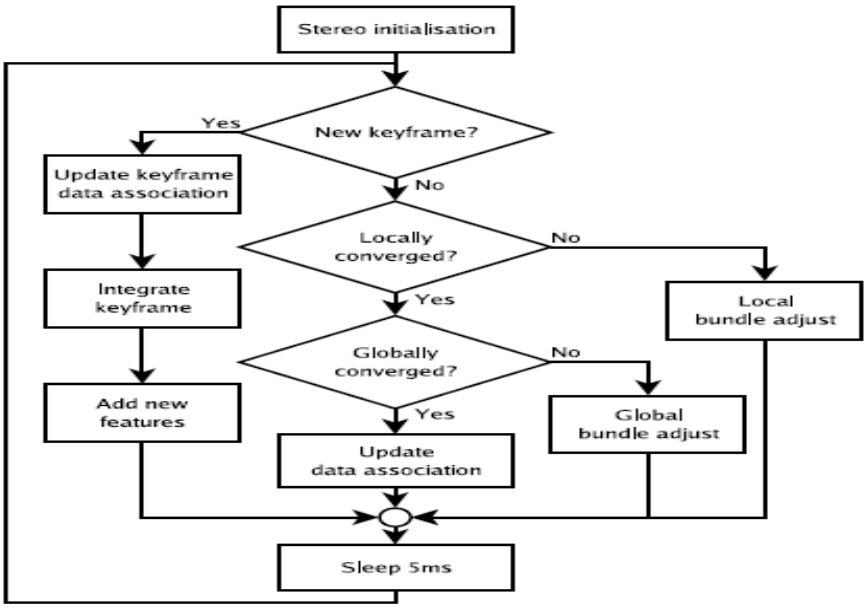
4. Pozicija kamere se ažurira na osnovu pronađenih korespondencija
5. Veći broj (cca 1000) točaka se re-projicira i traži na slici
6. Konačna pozicija se računa iz svih pronađenih odnosa

2.1.4. Mapiranje – izgradnja mape

U nastavku će okvirno biti objašnjeno kako se trodimenzionalne točke dodaju u koordinatni sustav. Koordinatni sustav se inicijalno izgrađuje uz pomoć stereo-algoritma u 5 točaka. Pri spomenutoj inicijalizaciji korisnik mora pritisnuti određenu tipku za početak praćenja. Zatim je potrebno kameru translatirati u jednu stranu, te otpustiti tipku. Spomenuti proces inicijalizacije na implementaciji sustava PTAM vizualno se može pratiti na slici 14. Nadalje, preslikavanje (mapiranje) se kontinuirano provodi sukladno dodavanjem novih ključnih slika. Ključne slike se dodaju ukoliko se uspostave sljedeći uvjeti: kvaliteta praćenja mora biti visoka, mora proći barem dvadeset slika (*frameova)* od generiranja prošle ključne slike i kamera mora biti minimalno udaljena od već poznate točke scene.

Procesom praćenja već su određeni rubovi. Zatim se koriste algoritmi za sužavanje skupa ovih rezultata. Preostale točke su kandidati za unošenje u koordinatni sustav. Pošto točka zahtjeva informaciju dubine, traži se najbliža ključna slika, te se triangulacijom pronalazi položaj točke.

Na slici 11. skiciran je proces preslikavanja - mapiranja.



Slika 11 - skica procesa preslikavanja

2.2. IMPLEMENTACIJA SUSTAVA PTAM

Gore opisani sustav implementiran je u C++ programskom jeziku. Kod je izvorno razvijen na x86-64 Linux operativnom sustavu. Također je uspješno prenesen i na MacOS (Intel-based; grafika X11). Program (softver) je uz nekakve preinake prenesen i na 32-bitnu Windows platformu.

2.2.1. Potrebna konfiguracija

Program zahtjeva minimalno dva procesa u isto vrijeme, te bi prema tome bio potreban više-jezgreni procesor. Osobno sam koristio Intel i5 2,4 GHz, međutim, razvojni tim garantira kvalitetne performanse i s Intel Core 2 Duo procesorom koji radi na istoj frekvenciji.

Program zahtjeva OpenGL grafički standard. Razvijen je i testiran isključivo za nVidia grafičku karticu i popratne pogonske programe. Prema tome, pri korištenju drugih GL pogonskih programa bile bi potrebne određene preinake u kodu.

Program se zasniva na ulaznom toku podataka, te zahtjeva video kameru s široko-kutnim objektivom rezolucije 640x480, te frekvencije 30 Hz. Iako razvojni tim preporučuje skuplje uređaje, osobno sam koristio Logitech Quickcam C100, uz prilične performanse i učinkovitost.

2.2.2. Potrebne biblioteke

Programsko ostvarenje ovog sustava zasniva se na tri glavne zavisnosti:

- TooN – glavna biblioteka za linearnu algebru
- libCVD – biblioteka za manipulaciju videom i slikama, te za računalni vid
- Gvars3 – run-time biblioteka za skriptiranje i konfiguraciju (potprojekt libCVD biblioteke)

Sve tri navedene biblioteke razvijene su u laboratoriju Cambidge Machine Intelligence i mogu se naći na [12] (biblioteka TooN) i na [13] (poveznica na biblioteke libCVD i Gvars3).

2.2.3. Instalacija

U navedenom će odjeljku biti objašnjena instalacija za Linux operacijski sustav. Pomoć pri instalaciji na Windows ili MacOS operacijskim sustavima potražite na [14].

Pri instalaciji, potrebno je nužne biblioteke instalirati sljedećim redoslijedom:

1. TooN; 2. libCVD; 3. Gvars3.

Sve se biblioteke instaliraju klasičnim **./configure; make; make install** protokolom.

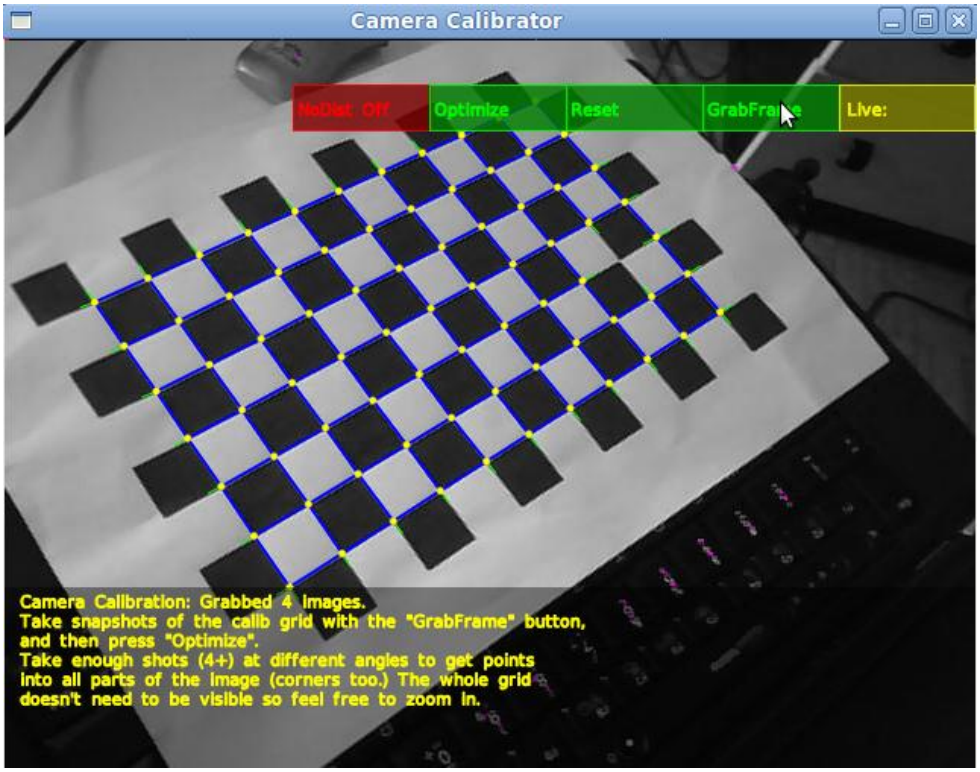
Zatim, valja kopirati datoteke iz direktorija **PTAM/Build/Linux** u direktorij **PTAM.**

Sljedeći korak je prevođenje (eng. compile) video postavki. Ponuđene su dvije datoteke: **patchVideoSource_Linux_DV.cc** i **VideoSource_Linux_V4L.cc**, koje rade s Unibrain Fire-i, odnosno Logitech QuickCam Pro 5000 kamerama. Meni osobno je odgovarala druga datoteka. Nadalje, treba prepraviti **Makefile** datoteku, prema kojoj će se željena datoteka prevesti. Potrebno je prepraviti samo vrijednost varijable **VIDEOSOURCE.**

Nadalje, cijeli se sustav prevodi uz pomoć naredbe **make;** te dobivamo dvije izvršne datoteke: **PTAM** i **CameraCalibrator.**

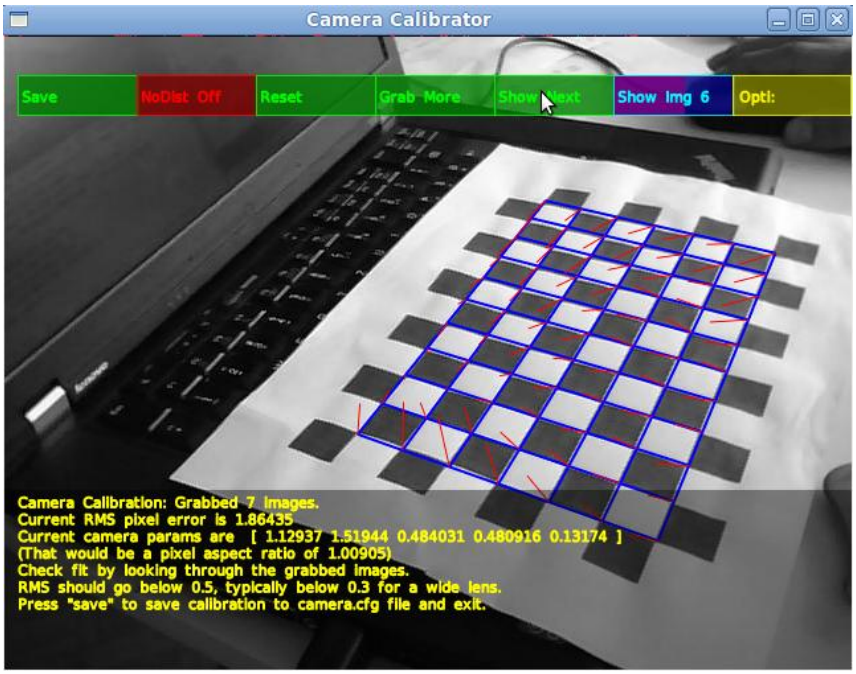
2.2.4. Pokretanje programa

Inicijaliziranje sustava obavljamo pokretanjem izvršne datoteke **CameraCalibrator**. U ovom se koraku od korisnika očekuje da kamerom snimi uzorak kao na slici 12. Ovim se postupkom sustav provjerava da li su sve tražene postavke uključene, te da li ulazni video tok uopće radi.



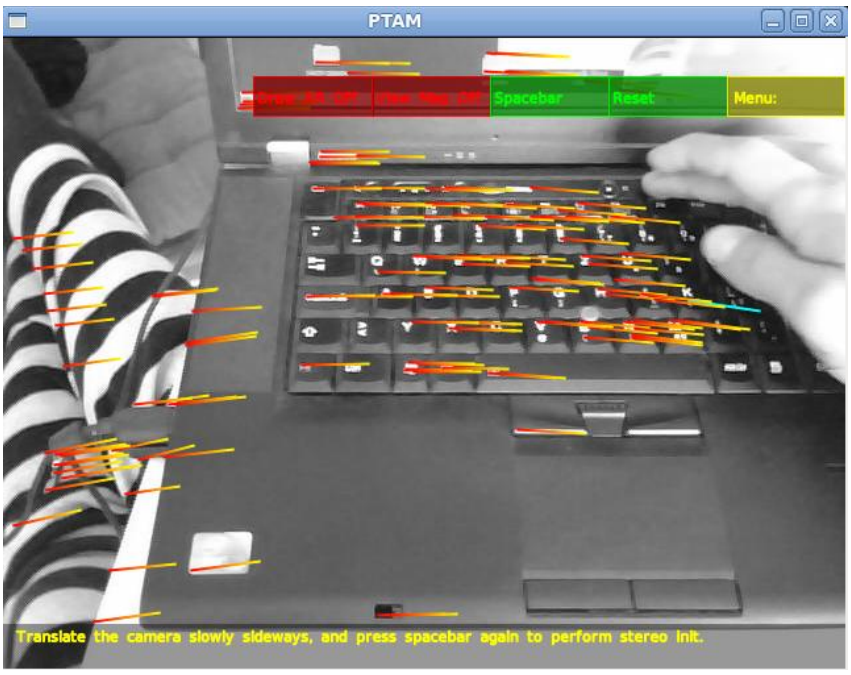
Slika 12 - Kalibracija kamere

Po uspješnom prepoznavanju uzorka, korisnik treba pritisnuti tipku „**GrabFrame“,** te isti postupak ponoviti u različitim pozama. Nakon par „uhvaćenih“ slika, potrebno je pritisnuti tipku „**Optimize“** kojom se kalkuliraju parametri kamere. Kad je korisnik zadovoljan s konvergencijom (greška RMS bi trebala biti oko 0.3 piksela, premda po vlastitom iskustvu to nije neophodan uvjet) potrebno je pritisnuti tipku „**Save“** kojom se spremaju postavke u **camera.cfg.** Opisani se postupak može pratiti na slikama 12 i 13.



Slika 13 - Završena kalibracija kamere

Po završetku kalibracije kamere, pokrećemo izvršnu datoteku **PTAM.** Na početku, sustav zahtjeva od korisnika izvršavanje inicijalizacije. Korisnik usmjeri kameru u željenu površinu (savršena bi bila ploha s izraženim detaljima – poput npr. tipkovnice).



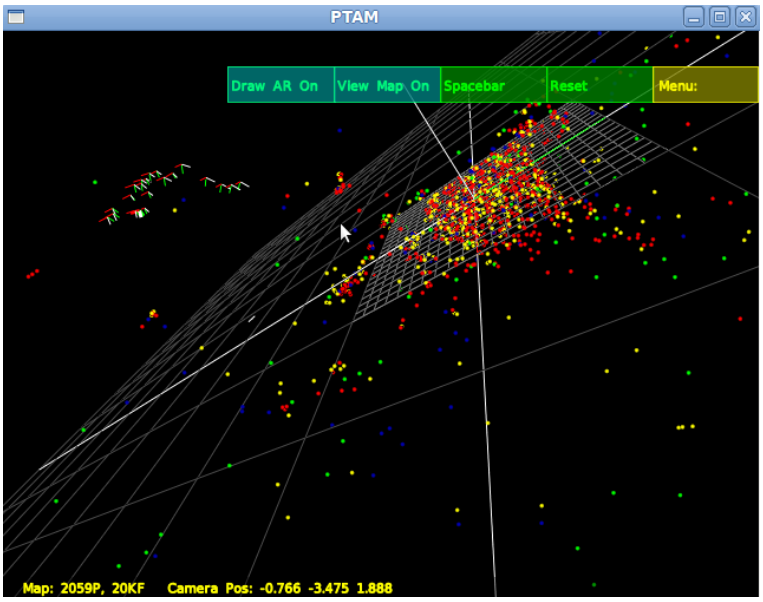
Slika 14 - Inicijalizacija mape

Nadalje, pritiskom na tipku razmaka (eng. spacebar), korisnik polako translatira kameru. Ponovnim pritiskom na prethodno navedenu tipku, ostvaruje se tzv. stereo par, iz kojeg sustav inicijalizira trodimenzionalni koordinatni sustav, te počinje proces praćenja. Spomenuti proces inicijalizacije primjećujemo na slici 14. Također je zgodno primijetiti na slici 15 kako inicijalna mapa sadrži vidno manje točaka od mape dobivene udaljavanjem/naknadnim praćenjem i preslikavanjem.

Slika 15 - Pronalazak površina i proširivanje mape

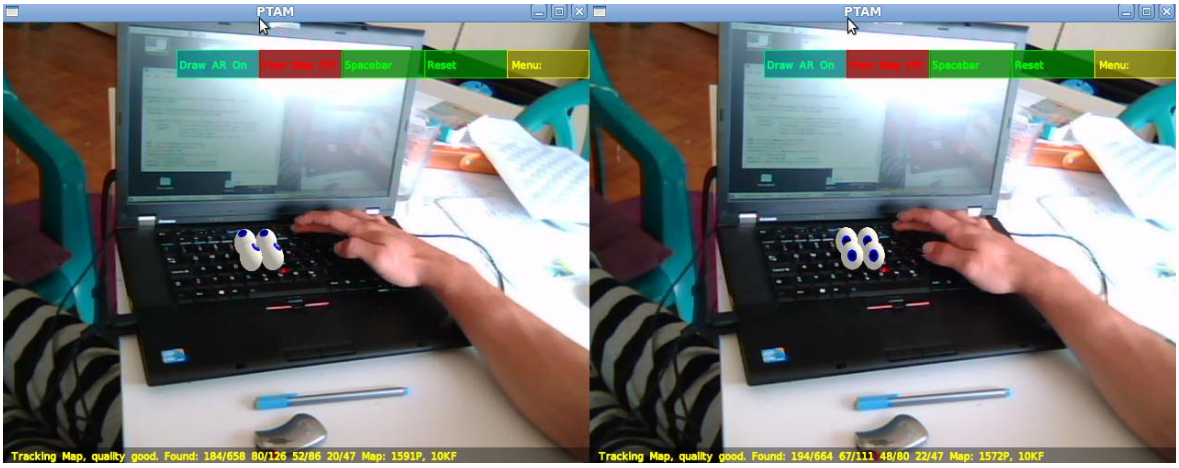
Jednostavna proširena stvarnost se može prikazati pritiskom na tipku „**Draw AR“.** Iscrtava se kvartet od po dva para očiju, koji nakon početne jednostavne animacije počnu pratiti kameru. Primjer iscrtavanja spomenute proširene stvarnosti vidi se na slici 17.

Ukoliko nije došlo do pravilnog generiranja koordinatnog sustava, potrebno je ponoviti proces inicijalizacije. Taj proces pokrećemo pritiskom na tipku **Reset,** te se nastavlja kao da je program tek pokrenut. Moguće da je do nepravilne inicijalizacije došlo zbog nedovoljno duge translacije. Također je bitno naglasiti da inicijalizacija samo s rotacijom neće omogućiti dovoljno dobru osnovu, te sustav neće moći nastaviti funkcionirati.



Slika 16 - Mapa - Koordinatni sustav

Potrebno je spomenuti da sustav pri praćenju i ažuriranju mape koristi samo crno-bijeli prikaz, te da se boja može fakultativno uključiti/isključiti.



Slika 17 - Iscrtavanje proširene stvarnosti

Razvojni tim napominje da je ostavljeno mnogo mjesta za napredak i ubrzanje algoritama praćenja. Međutim, sljedećim je mjerenjem pokazano da sustav uz 4000 točaka „odvaja“ svega nepunih 20 ms za praćenje jedne slike video toka podataka:

Priprema ključne slike	2.2 ms
Projekcija točaka	3.5 ms
Pretraživanje detalja	9.8 ms
Iterativno ažuriranje pozicije	3.7 ms
Ukupno	19.2 ms

Spomenuti je sustav ostvario i naknadne verzije. Par mjeseci po objavljivanju PTAM sustava ostvarena je i verzija PTAMM (eng. Parallel Tracking and Multiple Mapping) koja između sitnih poboljšanja nudi i permanentno spremanje mapa koja bi se pokretala ponovnim dolaskom/prepoznavanjem prethodno posjećenog mjesta. Više o ovom radu može se pronaći na [21].

3. ANDROID

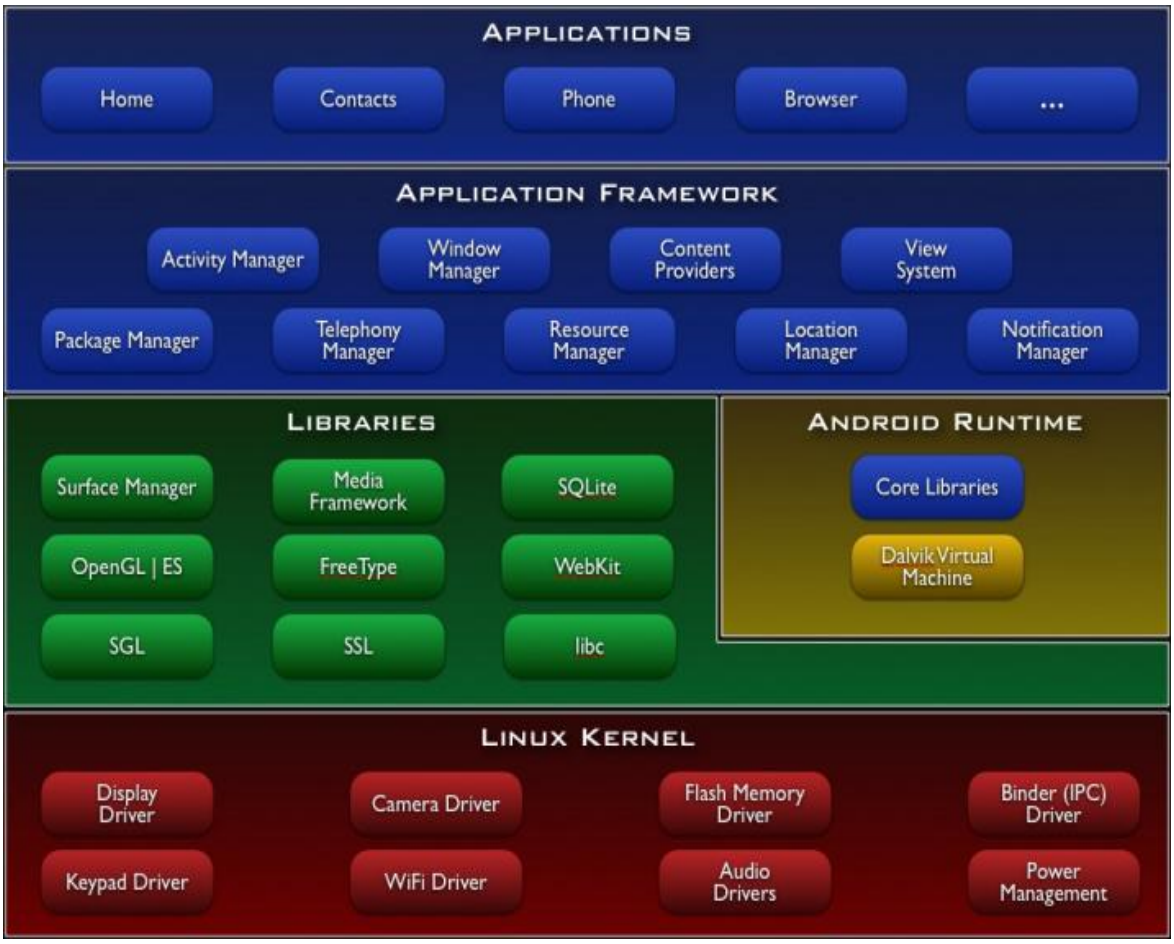
Pošto je glavna tema ovog diplomskog rada proširena stvarnost na ugrađenom sustavu, odabrao sam Android platformu kao svoju radnu okolinu. Ovo je poglavlje zamišljeno kao kratki opis Android platforme uz osvrt na razlike u odnosu na platforme prisutne na stolnim računalima.

Android je otvoreni sustav (eng. open source) iniciran od strane Google Inc. Baš zbog svoje „otvorenosti“ postavio se kao najlogičniji odabir. U svojoj osnovi, zamišljen je kao platforma za mobilne uređaje. Uključuje operacijski sustav, SDK (eng. Sofware Development Kit), programski radni okvir (eng. framework) i ključne aplikacije. Razvijen je od strane „Open Handset Alliance“ konzorcija koji se sastoji od 65 tehnologija i mobilnih kompanija, među kojima se nalazi prije spomenuti Google Inc., T-Mobile, HTC, Motorola i Qualcomm. Upravo posljednji nudi jedan zanimljiv okvir za izradu aplikacija iz domene proširene stvarnosti, koji će biti tema sljedećih poglavlja.

Najveći dio Android projekta objavljen je pod Apache 2.0 licencom (open source). To omogućuje svakome izgradnju željene verzije Androida. Primjerice, popularna

verzija CyanogenMod je modificirana verzija bazirana na Android 1.6, te pruža još mnoštvo mogućnosti koja su uključena u novije verzije Androida. Prema tome, ovakva licenca uvelike omogućava razvoj platforme od strane programera diljem svijeta.

Na slici 18 prikazana je arhitektura Android platforme. Platforma je bazirana na Linux jezgri (eng. kernel) verzije 2.6., koja omogućuje manipulaciju memorijom, model pogonskih programa, manipulaciju procesima i energijom/napajanjem, te sigurnosni model gornjim slojevima. Međutim, gornji su slojevi prilično izmijenjeni, te je nemoguće pokretati standardne Linux aplikacije na Android platformi.



Slika 18 - Arhitektura Android platforme

Povrh jezgre nalaze se nativne biblioteke, na slici 18 prikazane zelenim blokom. Android koristi mnoge postojeće *open-source* projekte za vlastitu platformu. Primjerice, biblioteka WebKit, koja se koristi za prikaz HTML stranica, koristi se i u web-pregledniku Safari. Tu se nalaze pretežito biblioteke za manipulaciju raznih medija i spremanje podataka. Među njima se nalazi i posebna verzija OpenGL-a

(OpenGL ES) o kojoj će kasnije biti riječi. Bibliotekama se pristupa putem Java aplikacijskog sučelja (API-ja).

3.1. ANDROID SDK

U ovom su poglavlju ukratko opisane komponente Android Software Development Kita potrebne za razvoj aplikacija opisanih u narednim poglavljima.

3.1.1. Dalvik VM (DVM)

Java je programski jezik koji se koristi za programiranje svih aplikacija na Androidu. Java aplikacije se generalno ne smatraju štedljive kad govorimo o memoriji. Na sustavima s više gigabajta to ne predstavlja veći problem. Međutim na mobitelima rijetko se kad susrećemo s viškom RAM-a. Također, niti snaga procesora nije najveća. Standardna se Java Virtualna Mašina (eng. Java Virtual Machine – JVM) ne susreće s navedenim problemima jer inicijalno nije stvorena za mobilne uređaje. Upravo zbog ovih činjenica, Android razvojni tim je razvio Virtualnu mašinu pod imenom Dalvik (DVM – eng. Dalvik Virtual Machine), dizajniranu za rad na uređajima koji rade na baterije sa sporijim procesorom i manje RAM-a.

Osnova ove virtualne mašine je Apache Harmony [10]. Java izvorni kod se inače prevodi u **class** datoteke, međutim, DVM ne može direktno pokretati ove datoteke. Umjesto toga, koristi se format tipa **dex.** Ovaj tip datoteke sadrži više klasa. Zgodno je napomenuti da ova vrsta datoteke ima manju veličinu od zbrojenih veličina svih **class** datoteka koje sadrži. Uzrok tome je što se konstantne vrijednosti i deklaracije metoda definiraju samo jednom.

Također, za razliku od JVM koji se bazira na stogu, DVM se bazira na registrima. Ovo je također jedan od načina uštede memorije.

Nadalje, standardni JVM obavlja kompilaciju *byte* koda u *machine* kod za vrijeme rada (eng. runtime), što ubrzava cijelu aplikaciju. DVM ne omogućuje spomenutu karakteristiku. Razlozi su sljedeći:

- Takav način prevođenja povećava korištenje memorije
- Mnoge su funkcionalnosti svejedno implementirane u nativnom kodu
- Mnoge su funkcionalnosti ostvarene hardverski, poput grafike i zvuka

Također, ekvivalent Javinoj **jar** izvršnoj datoteci na DVM je datoteka tipa **apk.** To je arhiva koja sadrži **dex** datoteke, resurse, nativne biblioteke i meta informacije. Ovo je datoteka koja se prenosi na mobilni uređaj prilikom instalacijskog procesa.

Android platforma susreće se i s nedostatkom FPU (eng. Floating Point Units) na mobilnim uređajima. Prema tome, operacije s decimalnim brojevima moraju se ostvarivati programerski. Po nekim procjenama takva ostvarenja računanja su otprilike 50 puta sporije od računanja s cijelim brojevima. Nadalje, u najvećem broju slučajeva, mobiteli ne raspolažu s više procesora, te nije omogućeno paralelno izvođenje programa. Uz smanjene hardverske performanse, ovo su razlozi koji ovelike otežavaju izvođenje programa.

3.1.2. Java API

Android službeno ne podržava niti Java SE niti Java ME. Međutim, dijelovi ovih API-ja su podržani na Androidu. Nadalje, Android pruža verziju svog API-ja za grafičko korisničko sučelje (GUI). Zbog toga sve klase **Swing** i **awt** biblioteka nedostaju. Najznačajniji nedostatak je ne podržavanje *Java beansa,* koji su sastavni dio velike većine *open source* projekata.

3.1.3. Grafički API

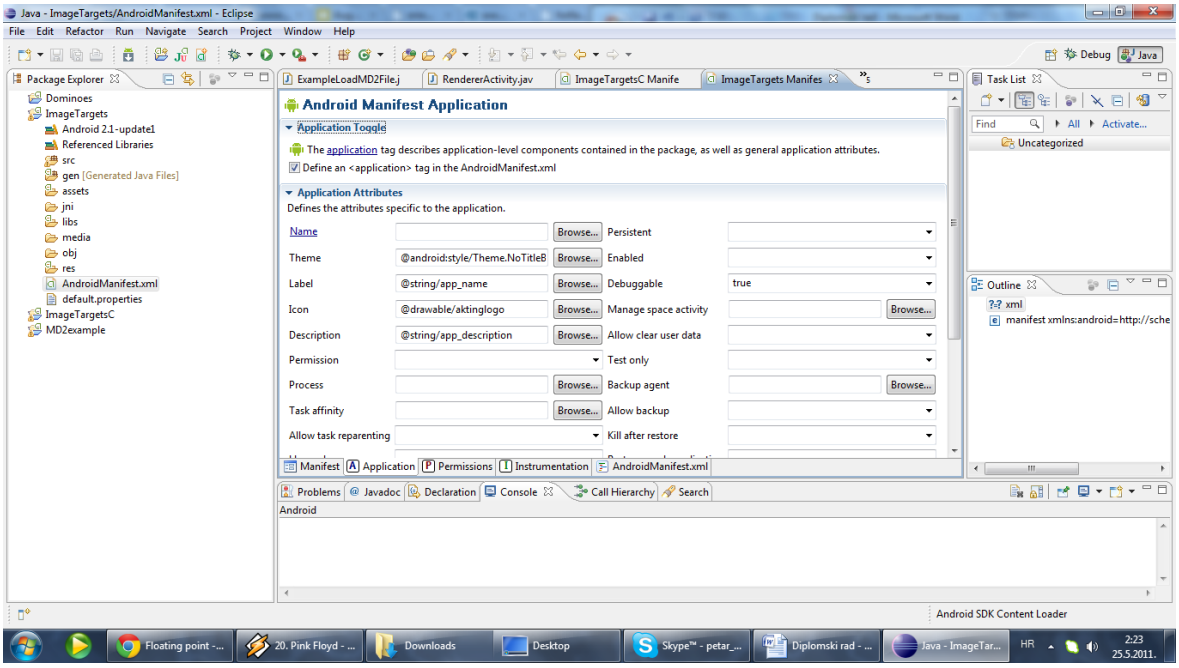
Android podržava OpenGL ES verziju OpenGL-a specijalno dizajniranu za ugrađene sustave. Ovaj 3D API je dostupan za Java i C++ programski jezik. Uključene su samo one mogućnosti OpenGL-a koje su stvarno potrebne. Zbog prethodno objašnjenog problema nedostatka FPU jedinice, na OpenGL ES 1.x se koristi računanje s fiksiranim decimalnim zarezom. OpenGL ES 2.0 koriste zapis s pomičnim zarezom (eng. floating point).

Najveća razlika u odnosu na standardni OpenGL je nedostatak **glBegin/glEnd** ulaznih točaka. Umjesto specificiranja svakog vrha, referenciraju se nizovi vrhova.

Sve teksture moraju biti kvadratne veličine potencije broja dva, kao u ranijim verzijama OpenGL-a.

3.1.4. Eclipse

Za razvoj aplikacija koristi se Eclipse SDK. Specifične funkcionalnosti dostupne su kroz Android SDK. Omogućeno je pokretanje na emulatoru ili direktno na mobilnom uređaju. Eclipse obavlja generiranje **apk** datoteke i instaliranje aplikacije na mobitel. Također, Eclipse omogućuje *debugg* kako na emulatoru, tako i na mobitelu.



Slika 19 - Eclipse radna okolina

3.1.5. Native Development Kit (NDK)

Java je jedini programski jezik koji je jedini podržan na Android platformi. Međutim, moguće je kombinirati Java kod s C++ kodom preko JNI (eng. Java Native Interface). NDK omogućuje korištenje klasa i metoda pisanih u nativnom jeziku (C/C++) preko dijeljenih biblioteka. Također, omogućeno je i korištenje Java klasa i metoda u nativnom kodu. Ovo omogućuje korištenje drugih biblioteka otvorenog koda (eng. open source) biblioteka, te pruža moguće ubrzanje ukoliko npr. iscrtavanje obavljamo preko nativnog koda. Spomenuti se alat koristi u programskom ostvarenju objašnjenom u slijedećem poglavlju.

4. PROŠIRENA STVARNOST NA ANDROID PLATFORMI

Kako je Android još uvijek mlada platforma, a proširena stvarnost prilično zahtjevna tehnologija, dostupno je svega par okvira (eng. framework) koji nude podršku proširene stvarnosti na Android ugrađenom sustavu. Također, ovi su sustavi u ranijim fazama razvoja. Jedan od njih je AndAR, sustav baziran na slavnom ARToolKit-u, biblioteci razvijenoj još 1999. godine.

Slika 20 - AndAR

Sustav AndAR neće biti tema užeg proučavanja ovog rada, te se detaljnije informacije o ovom projektu mogu pronaći na [16]. Predmet izučavanja ovog poglavlja biti će jedna drugi alat, QCAR. Ova je biblioteka, poput AndAR otvorene licence, i kao takva omogućava i olakšava razvoj proširene stvarnosti na Androidu.

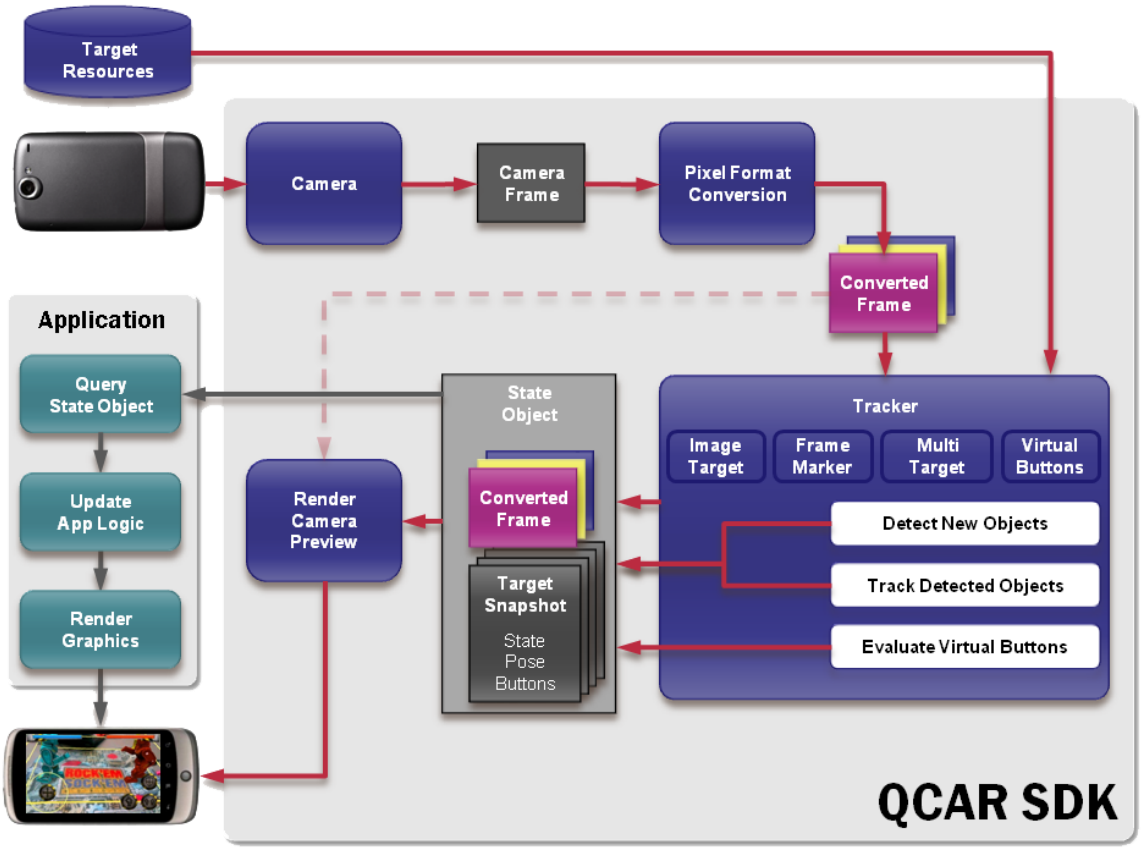
4.1. QCAR SDK

QCAR SDK je projekt razvijen od strane Qualcomm Inc., te objavljen u listopadu 2010. godine. QCAR uvelike olakšava razvoj aplikacija iz područja proširene stvarnosti. Baziran je na markerima dostupnim na stranicama projekta, o kojima će više govora biti u sljedećem poglavlju.

4.1.1. Arhitektura sustava

Aplikacija bazirana na QCAR SDK sastoji se od sljedećih komponenti. Na slici 21 prikazana je arhitektura sustava s imenima komponenti navedenim u zagradama:

- **Kamera (eng. Camera)**: Kamera odgovara predlošku vrste **singleton** koji osigurava da se u sustavu nalazi isključivo jedna instanca ove komponente. Kamera osigurava da se svaki *frame* uhvati i proslijedi *trackeru.* Uloga programera je samo pokretanje i zaustavljanje ove komponente. *Frame* se automatski dovodi u formatu i veličini ovisnoj o uređaju.
- **Pretvornik slika (eng. Image Converter)**: Ova komponenta pretvara sliku iz formata kamere (npr. YUV12) u format prikladan za iscrtavanje OpenGL-om (npr. RGB565) i praćenje. Također je uključeno i pod-uzorkovanje.
- **Tragač (eng. Tracker)**: Također **singleton,** te sadrži algoritme računalnog vida koji otkrivaju i prate objekte scene u svakoj slici video toka podataka. Također sadrži i algoritme koji otkrivaju markere i virtualne tipke. Rezultat se sprema u tzv. objekt stanja (eng. state object) koji se dalje koristi pri iscrtavanju, te je dostupan za manipulaciju iz programskog koda.
- **Pozadinski video pokazivač *(eng.* Video Background Renderer)*:*** Još jedna komponenta koja odgovara predlošku **singleton** koja je zadužena za iscrtavanje slika spremljenih u objekt stanja
- **Aplikacijski kod (eng. Application Code):** Programer mora inicijalizirati sve gore spomenute komponente i odraditi tri koraka u aplikacijskom kodu. Za svaki procesuirani slikovni okvir (eng. frame), ažurira se objekt stanja i poziva se metoda za iscrtavanje. Tri koraka su sljedeća:
    1. Upit objektu stanja za novo detektiranim ili novim stanjima prethodno detektiranih markera
    2. Ažuriranje aplikacijske logike s novim ulaznim podacima
    3. Iscrtavanje generiranih objekata - grafike
- **Markeri (eng. Target Resources):** Markeri se mogu generirati pomoću *on-line* sustava. Generiraju se posebne konfiguracijske datoteke koje omogućuju korisniku/programeru korištenje vlastitih markera. Konfiguracijske datoteke QCAR SDK koristi u vrijeme izvođenja.



Slika 21 - Arhitektura i tok podataka QCAR SDK u aplikacijskom okruženju

4.1.2. Aplikacijsko programsko sučelje - API

U ovom odlomku ukratko će biti spomenuti razredi QCAR biblioteke, te u par crta objašnjena njihova uloga. Za detaljniju dokumentaciju moguće je pritisnuti na naziv svakoga od razreda.

QCAR::Area – osnovni razred za 2D oblike korištene u QCAR-u

QCAR::CameraCalibration – utvrđuje unutrašnje parametre kamere

QCAR::CameraDevice – implementira pristup ugrađenoj kameri mobilnog uređaja

QCAR::Frame – kolekcija različitih prikaza jedne slike video toka podataka. Može sadržavati proizvoljan broj slikovnih prikaza u različitim formatima ili rezolucijama zajedno s vremenskom oznakom i indeksom

QCAR::Image – jedna slika. Npr. proslijeđena s kamere

QCAR::ImageTarget – ciljana ravnina za iscrtavanje. Svako manipuliranje s virtualnim tipkama obavlja se preko ovog razreda

QCAR::Marker – pravokutni marker

QCAR::Matrix34F – matrica s 3 reda i 4 stupca decimalnih brojeva

QCAR::Matrix44F – matrica s 4 reda i 4 stupca decimalnih brojeva

QCAR::MultiTarget – skup ciljanih ravnina za iscrtavanje s fiksiranom prostornom udaljenosti

QCAR::NonCopyable – osnovni razred za sve objekte koji se ne mogu kopirati. Među njima su CameraCalibration, CameraDevice, Image, Renderer, Trackable, Tracker, VirtualButton i svi njihovi podrazredi

QCAR::Rectangle – definira dvodimenzionalni pravokutni prostor

QCAR::Renderer – pruža metode za iscrtavanje video pozadine i trodimenzionalnih objekata na osnovu podataka o poziciji

QCAR::State – jedan od bitnijih razreda. Pruža konzistentan pogled na stanje proširene stvarnosti uključujući i *frame* i sve instance razreda *Trackable.*

QCAR::Trackable – predstavlja osnovni razred za sve objekte koji mogu biti praćeni u 6DOF (eng. Six Degrees of Freedom – translatacija i rotacija u svim smjerovima)

QCAR::Tracker – razred zadužen za praćenje i manipulaciju svim *Trackable* objektima

QCAR::UpdateCallback – služi za ažuriranje čitavog sustava

QCAR::Vec2F – dvodimenzionalni vektor s decimalnim brojevima

QCAR::Vec2I – dvodimenzionalni vektor s cijelim brojevima

QCAR::Vec3F – trodimenzionalni vektor s decimalnim brojevima

QCAR::Vec3I – trodimenzionalni vektor s cijelim brojevima

QCAR::Vec4F – četverodimenzionalni vektor s decimalnim brojevima

QCAR::Vec4I – četverodimenzionalni vektor s cijelim brojevima

QCAR::VideoBackgroundConfig – čuva konfiguraciju pozadinskog videa

QCAR::VideoMode – implementira pristup ugrađenoj kameri uređaja

QCAR::VirtualButton – virtualna tipka na površini koja se može pratiti.

4.1.3. Android dozvole

Svaka aplikacija na Android platformi sadrži konfiguracijsku datoteku **AndroidManifest.xml** u svom osnovnom direktoriju. Manifest operacijskom sustavu prikazuje osnovne informacije o aplikaciji prije svakog pokretanja aplikacije. Među informacija o raznim imenima komponenti aplikacije, uključenih biblioteka, minimalne verzije Android sustava itd., nalaze se i dozvole za korištenje određenih komponenti uređaja i samog operacijskog sustava. Tako za pokretanje QCAR SDK aplikacije neophodno je omogućiti sljedeće dozvole (Android Permissions):

- android.permission.ACCESS_NETWORK_STATE
- android.permission.ACCESS_WIFI_STATE
- android.permission.CAMERA
- android.permission.INTERNET
- android.permission.READ_PHONE_STATE

4.1.4. Markeri – „Trackables“

„Trackables“ predstavlja osnovi razred koji predstavlja sve objekte u stvarnom svijetu koje QCAR SDK može pratiti u šest stupnjeva slobode. Svaka instanca tog razreda može biti prepoznata i praćena, ima ime, identifikacijski broj, status, te informacije o poziciji. Razredi „Image Targets“, „Multi Targets“ i „Markers“ su podrazredi, te nasljeđuju sve atribute od „Trackables“ osnovnog razreda. Instance ovog razreda (i instance njegovih podrazreda) se ažuriraju svakog okvira video toka podataka, te se u tzv. statičkom objektu (eng. Static Object) prenose u samu aplikaciju.

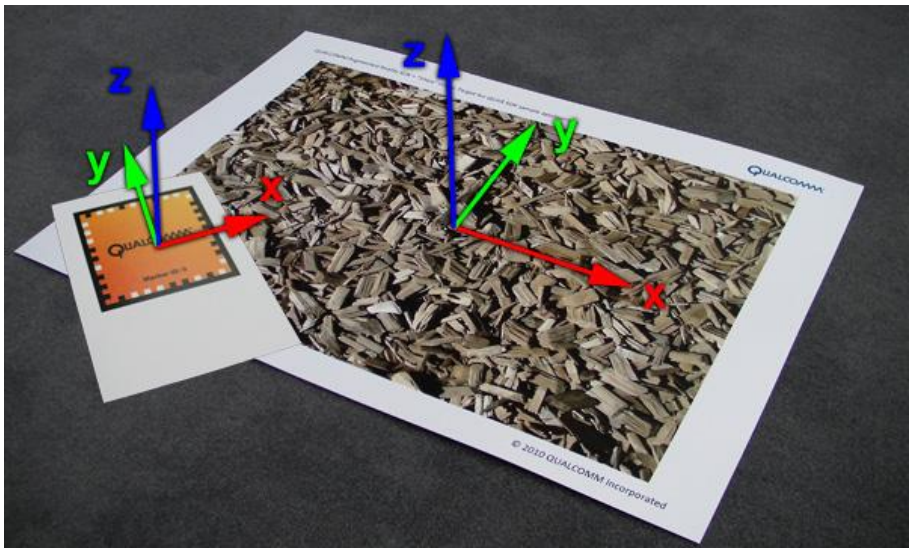
Četiri su tipa ovog razreda, te su označeni sljedećim enumeratorima:

- UNKNOWN_TYPE - *Trackable* nepoznatog tipa.
- IMAGE_TARGET - *Trackable* tipa *ImageTarget*
- MULTI_TARGET - *Trackable* tipa *MultiTarget*
- MARKER - *Trackable* tipa *Marker*

Nadalje, sljedećim se enumeratorima označava status svake instance razreda *Trackable:*

- UNKNOWN – stanje je nepoznato. Ovo stanje se najčešće vraća prije inicijalizacije.
- UNDEFINED – stanje nije definirano.
- NOT_FOUND – *Trackable* nije pronađen. Do ovog stanja dolazi npr. ukoliko referencirani *Trackable* nije dio baze podataka.
- DETECTED – *Trackable* je pronađen u okviru video toka podataka.
- TRACKED – obavljeno je praćenje na *Trackableu* u okviru video toka podataka.

Ukoliko je *Trackable* pronađen i praćen, dalje se procesuira pozicija u obliku matrice 3x4. Na slici 22 možemo vidjeti pravilno detektirane i praćene dvije instance razreda *Trackable.*



Slika 22 – Trackable – marker pomoću kojeg se definira položaj u okolini

4.1.4.1. *Image Targets*

Ovaj podrazred *Trackable* razreda nas najviše zanima u ovom radu. Radi se o najjednostavnijoj, međutim za potrebe ovog rada sasvim dovoljnoj implementaciji razreda *Trackable.* U praksi, radi se o običnom markeru, slici na papiru koju QCAR SDK prepoznaje, te na osnovu pozicije omogućava iscrtavanje proširene stvarnosti na ekran.

U instalacijskom paketu QCAR SDK dobiva se pet pokaznih primjera aplikacija s kojima dolaze i korespondentni markeri/*Trackables. Trackable* markeri se definiraju u datoteci **assets/config.xml.** Generiranje proizvoljnih markera omogućeno je na [17]. Slanjem slike formata JPG ili PNG, dobivaju se povratne datoteke, među kojima i prethodno navedena XML datoteka, koje je potrebno uključiti u projekt. Na slici 23 vidimo naš *Trackable* i njegove referentne detalje na osnovu kojih se prepoznaje u sceni.

Slika 23 - FER *Trackable* (lijevo) i njegovi referentni detalji (desno)

4.1.5. Virtualne tipke - Virtual Buttons

Ovim razredom omogućena je interakcija s proširenom stvarnosti u aplikaciji baziranoj na QCAR SDK. To je prethodno definirana površina na *Image Targetu,* koja dodirom ili prekrivanjem pokreće događaj (eng. event). Ovaj razred se može koristiti za razne implementacije: od klasičnog pritiskanja tipaka, detektiranja da li je određeni dio *Image Targeta* prekriven, pa sve do raznih interakcija s modelima

na sceni proširene stvarnosti. Virtualne tipke se evaluiraju jedino ako su unutar dosega kamere, te je video tok podataka postojan određeno vrijeme. Evaluacija Virtualnih tipki je isključena prilikom brzih pokreta kamerom.

Virtualne tipke se mogu prethodno dodati u **config.xml** datoteku kao segment *Image Targetu*, te se mogu dinamički dodavati prilikom izvođenja aplikacije.

Sljedeći su najvažniji parametri Virtualne tipke:

- Naziv / identifikator
- Koordinate: Virtualne tipke su definirane kao pravokutne površine. Definiraju se gornji-lijevi i donji-desni kut pravokutnika. Potrebno je naglasiti da se za jedinice za definiranje koordinata definiraju u lokalnom koordinatnom sustavu. Izvor koordinatnog sustava je u sredini roditeljskog *Image Targeta*
- Osjetljivost: - HIGH – brza detekcija, može prouzročiti netočne aktivacije
  - MEDIUM – najčešće optimalna detekcija
  - LOW – potrebno je duže prekriti tipku za pokretanje događaja
- Događaji – korespondentne akcije koje se poduzimaju osnovu prekrivanja, odnosno otkrivanja Virtualne tipke

4.2. IMPLEMENTACIJA SUSTAVA QCAR SDK

4.2.1. Instalacija sustava QCAR SDK

U sljedećem poglavlju bit će ukratko objašnjen postupak instalacije za Windows operacijski sustav. Pri instalaciji za Linux sustav postupak je sličan, međutim komponenta Cygwin nije potrebna, jer je postupak izgradnje (eng. build) omogućen u operacijskom sustavu. Za potpunu instalaciju te razvoj aplikacija baziranih na QCAR SDK potrebne su sljedeće komponente. Pri instalaciji komponenti je potrebno instalirati navedenim redoslijedom:

1. JDK (eng. Java SE Development Kit) – omogućuje programiranje u programskom jeziku Java i sadrži osnovne Java biblioteke
2. Eclipse IDE (eng. Integrated Development Environment) – okolina koja omogućava programiranja aplikacija, njihovo prevođenje i pokretanje
3. Android SDK (eng. Software Development Kit) – omogućuje programiranje Android programskih aplikacija i korištenje osnovnih biblioteka
4. Android ADT (eng. Android Development Tools) – alat/dodatak za Eclipse koji omogućuje brže i lakše razvijanje i distribuiranje aplikacija
5. Android SDK podrška za platforme (eng. Android SDK Platform Support) – pomoću aplikacije **Android SDK and AVD Manager** omogućuje ažuriranje i skidanje novih komponenti, pokretačkih programa za mobilne uređaje, te generiranje virtualnih uređaja za pokretanje Android aplikacija
6. Cygwin okruženje – okruženje koje omogućuje prevođenje nativnih klasa
7. Android NDK (eng. Native Development Kit) – ekstenzija na Android SDK koja omogućuje izgradnju dijelova programskog koda u nativnom jeziku. Nativni se jezici koriste za programske kodove kritičnih performansi (npr. iscrtavanje). U našoj aplikaciji koristi se C++ programski jezik.
8. QCAR SDK

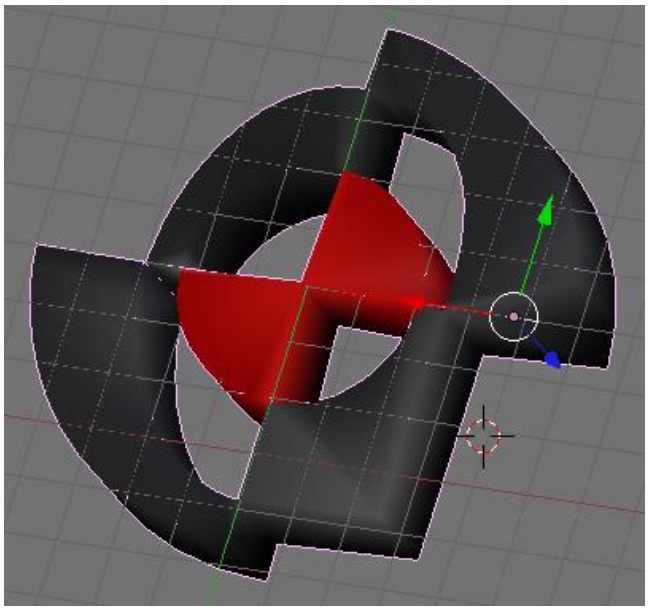
Nadalje, potrebno je postaviti globalnu varijablu QCAR_SDK_ROOT na vrijednost direktorija u koji je instaliran QCAR SDK. Npr. „C:/Development/Android/qcar-sdk-xx-yy-zz“, gdje su xx-yy-zz vrijednosti korespondentne verzije SDK-a. Poveznice na navedene programske komponente nalaze se na [17].

4.2.2. Implementacija Virtualnih tipki

QCAR SDK s instalacijom pruža pet primjera koji omogućuju brže shvaćanje i razvoj vlastitih aplikacija. U ovom poglavlju objašnjena je integracija vlastitih modela te upotreba Virtualnih tipki za mijenjanje tekstura modela. Navedeni je postupak implementacije baziran na primjeru *Virtual Buttons,* uz određene preinake.

Za početak nam je potreban primjeren *Image Target.* Ovaj je postupak objašnjen u poglavlju 4.1.4.1. Bitno je navesti da nam je za upotrebu Virtualnih tipki potreban dovoljno detaljan *Image Target.* Razlog tome je što se pri detektiranju Virtualnih tipki koriste detalji dostupni na *Image Targetu.* Ukoliko „ispod“ Virtualne tipke nema detalja, nemoguće je detektirati prekrivanje/otkrivanje tipke.

Nadalje, potreban nam je model koji ćemo iscrtavati. Za potrebe ovog primjera, izradio sam jednostavan model prikazan na slici 24.



Slika 24 - 3D model

Za iscrtavanje modela u aplikaciji baziranoj na QCAR SDK potreban nam je **.h** *header* format. Prema vlastitom iskustvu, do traženog formata je najlakše doći izvozom željenog modela u **.obj** formatu, te konverzijom pomoću skripte dostupne na [18]. Potrebno je spomenuti da ovim pristupom nije moguće implementirati animaciju modela. Za animiranje potrebno je koristiti nekakav programski okvir koji omogućuje prikaz i animaciju (eng. rendering framework) baziran na Android platformi i OpenGL ES-u. Međutim, pošto je platforma još „mlada“, takvi su programski okviri pretežito u testnim fazama. Također, bitno je naglasiti da je iscrtavanje poželjno odraditi u nativnom kodu, zbog skupoće samog procesa iscrtavanja.

Nadalje, potrebno je željeni model integrirati u segment aplikacije zadužen za iscrtavanje. U korištenom primjeru radi se o **VirtualButtons.cpp** razredu dostupnom među datotekama nativnog koda u direktoriju **jni/.** Pošto naš

generirani model ne sadrži polje indeksa (eng. indeks array), potrebno je koristiti metodu **glDrawArrays** umjesto **glDrawElements.** Nadalje, potrebno je refaktorirati i izmijenjati kod baziran na ovim metodama.

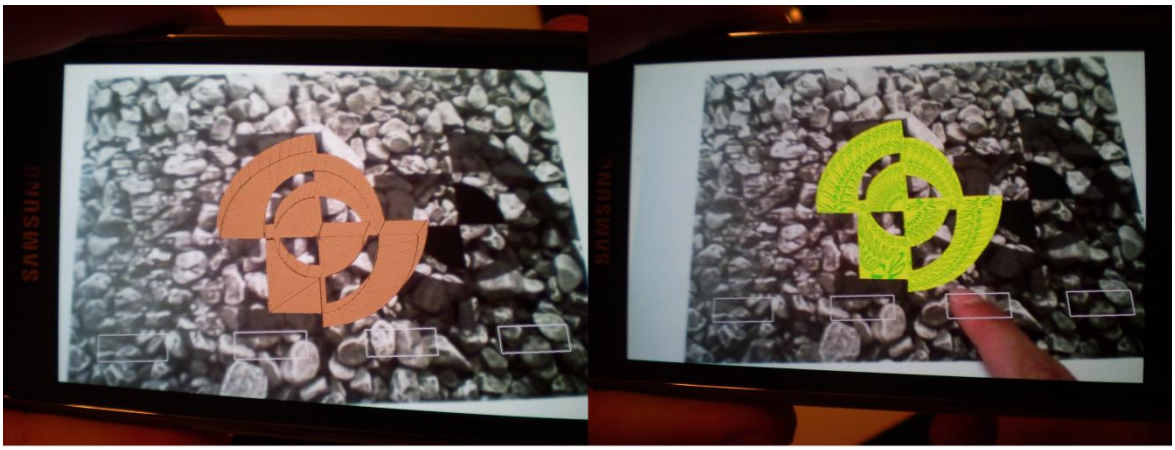
Sama inicijalizacija Virtualnih tipki je prilično dobro obavljena u primjeru, te ćemo je velikim dijelom koristiti za naš program. Prema tome, potrebno je implementirati akcije koje se provode po aktiviranju Virtualnih tipki. U našem primjeru, na osnovu pritisaka na različite tipke, naš će se model „zavijati“ u različite teksture. Pošto je naglasak ovog poglavlja samo prikazati osnove funkcionalnosti QCAR SDK, navedene su teksture jednostavne slike u boji.

Iteriranjem kroz dostupne Virtualne tipke, svakim okvirom video toka podataka potrebno je provjeriti da li i koja je tipka pritisnuta. Na osnovu toga, za korištenu se odabire korespondentna tekstura. Također, za svaku od dostupnih tipki se iscrtavaju okviri na ekranu, što može biti praktična funkcionalnost.

Nadalje, potrebne su preinake u Java programskom kodu. Ove se preinake prvenstveno odnose na učitavanje željenih tekstura.

Prije pokretanja same aplikacije potrebno je prevesti sav kod u nativnom jeziku. Za taj postupak koristimo skriptu **ndk-build** prethodno spomenutog alata Android NDK.

Na slici 25 može se vidjeti primjer iscrtavanja i korištenja Virtualnih tipki za promjenu teksture na modelu.



Slika 25 - Virtualne tipke

4.3. QCAR + UNITY

QCAR ekstenzija za alat Unity [19] omogućuje integraciju QCAR funkcionalnosti u Unity 3 IDE i time omogućuje brži razvoj aplikacija i igara. U ovom je poglavlju ukratko opisan sam alat Unity, ekstenzija, te je na jednom primjeru objašnjena implementacija jednostavnog sustava.

4.3.1. Unity

Unity je integrirani alat za kreiranje 3D video igrica ili drugog interaktivnog sadržaja poput stvarno-vremenskih 3D animacija. Omogućuje razvoj aplikacija prvenstveno temeljeno na grafičkom okruženju, što znači da se čitava scena može izgraditi doslovno „povlačenjem“ sadržaja u okolinu.

Slika 26 - Unity

Alat je dostupan za Windows i Mac OS X platforme, te je pomoću njega moguće razvijati aplikacije za sljedeće platforme:

- Windows
- Mac
- Linux (trenutno u razvoju)
- Wii
- iPad
- iPhone
- Android

Također, moguće je razvijati i aplikacije za preglednike, uz koje je dostupan i dodatak „Unity web player plugin“. Podrške za Xbox 360 i PlayStation 3 su u finalnoj fazi razvoja. Prema tome, radi se o iznimno kvalitetnom i široko korištenom alatu.

Unity je dostupan u dvije glavne licencne verzije: *Unity* i *Unity Pro.* Osnovna je verzija besplatna, dok je *Pro* verzija dostupna za određenu cijenu. U *Pro* verziji dostupne su mnoge funkcionalnosti, dok se u besplatnoj verziji iscrtava vodeni žig (za web aplikacije), odnosno *splash screen* (za ostale – samostalne aplikacije).

Grafički pogon (eng. engine) koristi Direct3D (Windows), OpenGL (Mac, Windows), te OpenGL ES (iPhone OS, Android). Za skriptiranje samih komponenti scene koriste se *UnityScript* (skriptni jezik baziran na JavaScriptu), C# i *Boo* (skriptni jezik baziran na Pythonu).

Podržana je integracija modela iz svih popularnijih alata za modeliranje: 3ds Max, Maya, Blender, Modo, ZBrush, Cinema 4D i Cheetah3D.

Unity podržava dugi niz funkcionalnosti i kompatibilnosti, međutim, za potrebe naše aplikacije ćemo koristiti samo uži niz komponenti.

4.3.2. QCAR Unity Extension

Na osnovu ovog dodatka moguće je koristiti komponente oba sustava. Moguće je koristiti komponente QCAR-a, poput Virtualnih tipki, *Image Targeta* (koji se koristi za kao referentni marker) i AR kamere (koja implementira cijelu funkcionalnost praćenja i detektiranja prostora za iscrtavanje. S druge strane, sve navedene komponente mogu se koristiti u jednoj jednostavnoj razvojnoj okolini poput Unityja.

Prema tome, razvoj aplikacije koja omogućava proširenu stvarnost, pa čak i interakciju s modelima, svodi se na slaganje komponenata na scenu. Naravno, svako ponašanje određene komponente potrebno je programirati u skriptama te „pridjenuti“ željenoj komponenti. Razvoj jedne takve aplikacije objašnjeno je u sljedećem poglavlju.

4.4. IMPLEMENTACIJA SUSTAVA QCAR UNITY EXTENSION

4.4.1. Instalacija potrebnih komponenti

U sljedećem poglavlju bit će ukratko objašnjen instalacijski postupak za Windows platformu. Inačica za Linux operacijske sustave još nije dostupna u *release* verziji, dok se instalacija za Mac OS X identična Windows instalaciji.

Za razvoj aplikacija temeljenih na QCAR Unity Extensionu potrebne su nam dvije komponente:

- Unity alat verzije 3.2. ili 3.3., dostupan na [19]
- QCAR Unity Extension, dostupan na [17]

Po skidanju i instalaciji Unity alata, potrebno je skinuti ekstenziju te pokrenuti izvršnu datoteku. Nadalje, odabire se željena lokacija za razvoj aplikacija. Paketi se automatski kopiraju u **Standard Package** instalacijskog direktorija Unity alata.

Ukoliko se koristi 64-bitni operacijski sustav potrebno je skinuti i instalirati dodatak na [20] za osiguravanje stabilnosti cijelog sustava.

Rezultat instalacije će biti sljedećih pet Unity paketa:

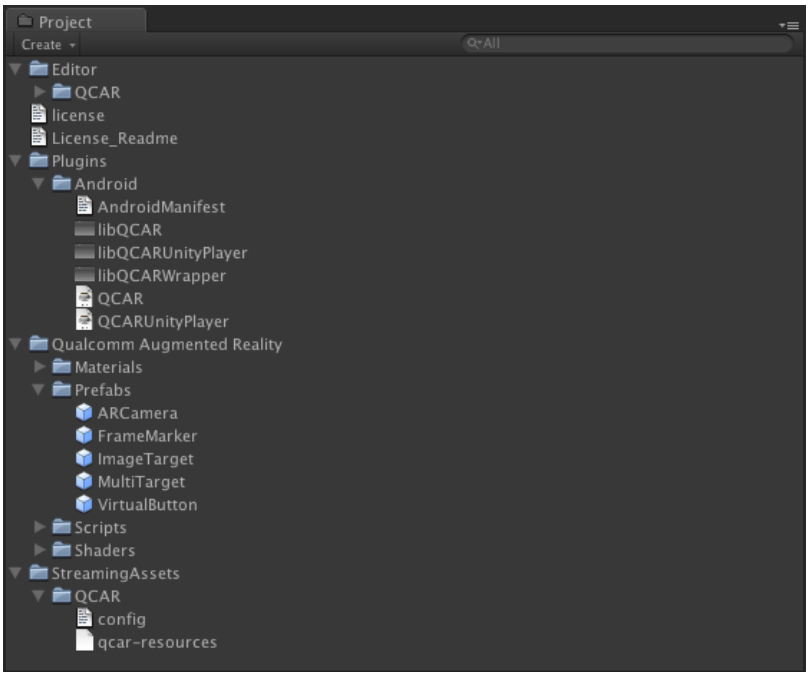
- **QCAR-1.0.0.unitypackage**: osnovna QCAR ekstenzija
- **QCAR-ImageTargets-1.0.0.unitypackage**: primjer koji koristi *Image Target* marker
- **QCAR-FrameMarkers-1.0.0.unitypackage**: primjer korištenja *Frame Markers* markera
- **QCAR-MultiTargets-1.0.0.unitypackage**: primjer korištenja *Multi Targets* markera
- **QCAR-VirtualButtons-1.0.0.unitypackage**: primjer koji koristi Virtualne tipke – *Virtual Buttons*

4.4.2. Implementacija jednostavnog projekta

Po instalaciji QCAR paketa, kreira se novi projekt. Potrebno je odabrati paket **QCAR-1.0.unitypackage** iz liste izbornika **Import the following packages.** Alternativno, moguće je uvesti paket desnim klikom na pogled **Project,** te odabirom izbornika **Import Package.**

Nadalje, za korištenje markera u aplikaciji, dodaju se konfiguracijske datoteke **config.xml** i **qcar-resources.dat** u direktorij **StreamingAssets/QCAR.** U poglavlju 4.1.4.1. objašnjeno je generiranje vlastitih markera, odnosno moguće je koristiti dostupne markere preuzimanjem konfiguracijskih datoteka iz primjera **QCAR-ImageTargets.**

Struktura direktorija i datoteka bi trebala izgledati kao na slici 27.



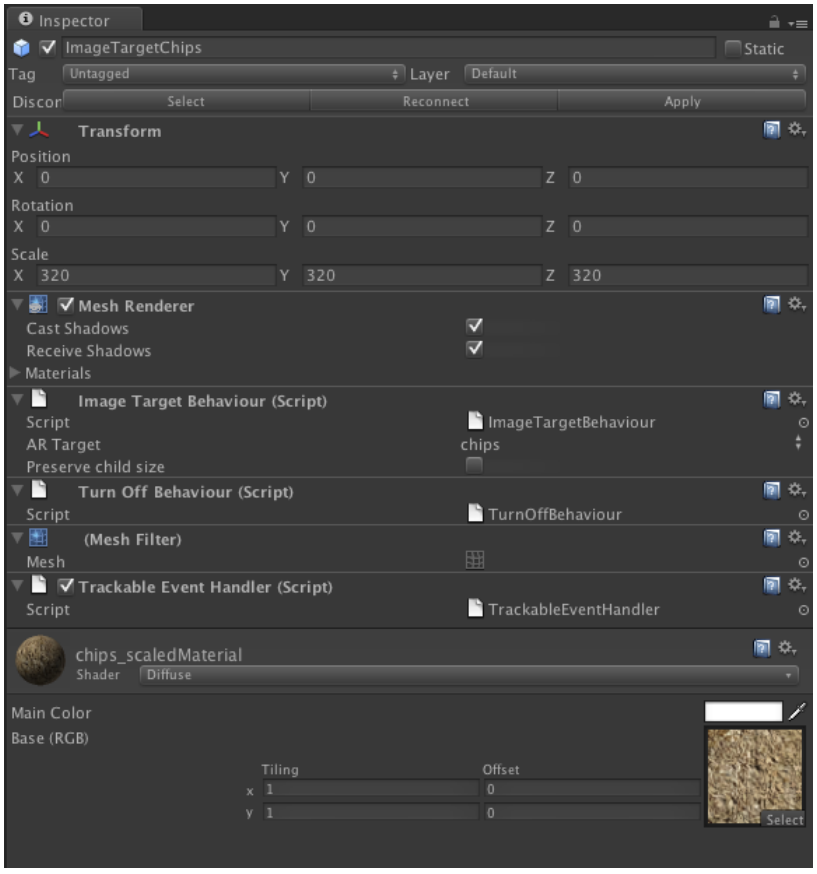
Slika 27 - Unity: Struktura podataka jednostavnog QCAR projekta

U nastavku je ukratko objašnjen sadržaj svakog od direktorija:

- **Editor:** sadrži skripte potrebne za dinamičku interakciju s elementima koji se mogu pratiti na sceni
- **Plugins:** sadrži Java i nativne binarne datoteke koje integriraju QCAR SDK u Unity Android aplikaciju

- **Qualcomm Augmented Reality:** sadrži skripte i tzv. *prefab* komponente. *Prefab* komponente su objekti specifični za Unity okolinu, koji se mogu višekratno koristiti (eng. reusable), pomoću kojih se gradi scena. U ovom direktoriju nalaze se navedeni objekti pomoću kojih se ostvaruju funkcionalnosti neophodne za proširenu stvarnost
- **Streaming Assets:** sadrži prethodno navedene datoteke **config.xml** i **qcar-resources.dat**

Sada kada su uključeni potrebni paketi u naš projekt, relativno je jednostavno slagati željenu scenu. U direktoriju **Qualcomm Augmented Reality/Prefabs** potrebno je označiti komponentu **ARCamera,** te jednostavnim povlačenjem u prozor **Scene** integrirati u scenu. Jednako tako dodaju se sve željene komponente. Neophodno je dodati i komponentu **ImageTarget** koja omogućuje korištenje markera, te samim time i praćenje.



Slika 28 - *Inspector View*

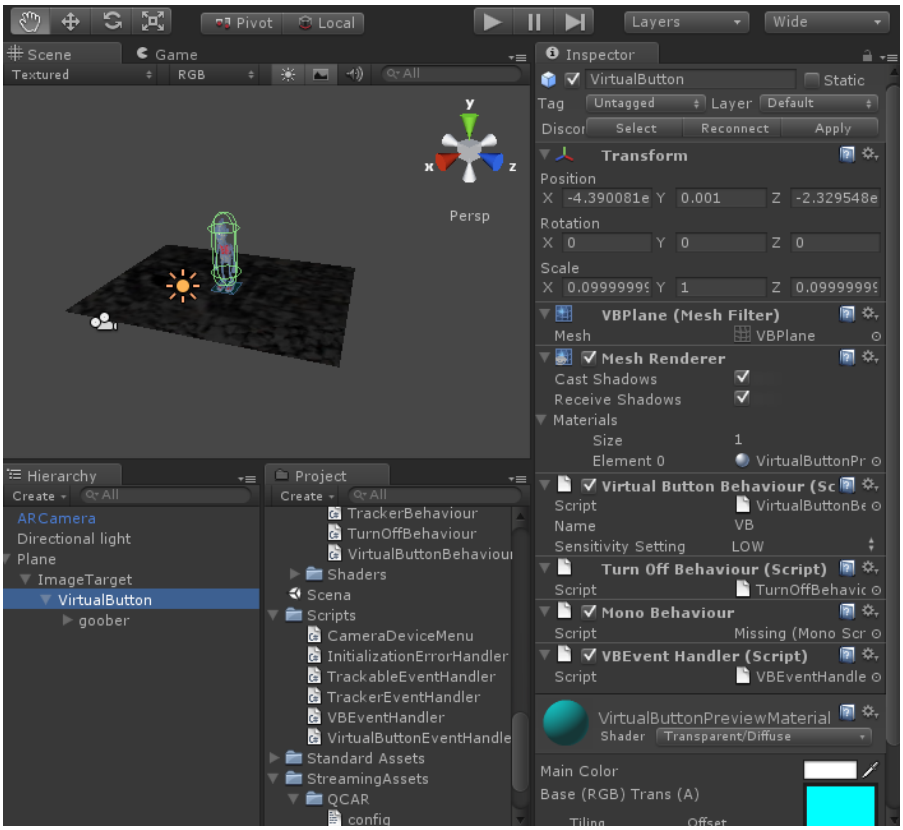
Za dodavanje proizvoljnih elemenata u scenu, poput modela, potrebno je prethodno dodati željene elemente u projekt, te ih onda „povući“ u scenu.

Jednostavni generički modeli poput osnovnih geometrijskih likova, svijetla, GUI elemenata, itd. mogu se dodati odabirom izbornika **GameObject->Create Other.**

Nadalje, svakom se elementu scene mogu dodavati skripte na osnovu kojih se ta komponenta scene ponaša. Tako je, između ostalog, uz element scene *ImageTarget* uključena skripta **Image Target Behavior,** s kojom se definira marker koji će se koristiti, te se definiraju i inicijaliziraju elementi sadržani na *ImageTarget* komponenti, poput Virtualnih tipki. Na slici 28 moguće je vidjeti sadržaj pogleda *Inspector* na kojem definiramo osnovne izgledne i ponašajne atribute elemenata projekta.

4.4.2.1. Aplikacija – „Interaktivni virtualni patuljak“

Primjer opisan u nastavku iscrtava jedan kinematički model patuljka na definiranom markeru, te omogućuje jednostavnu interakciju. Tako, primjerice, ukoliko prstom prekrijete prostor na kojem patuljak stoji, patuljak će skočiti.

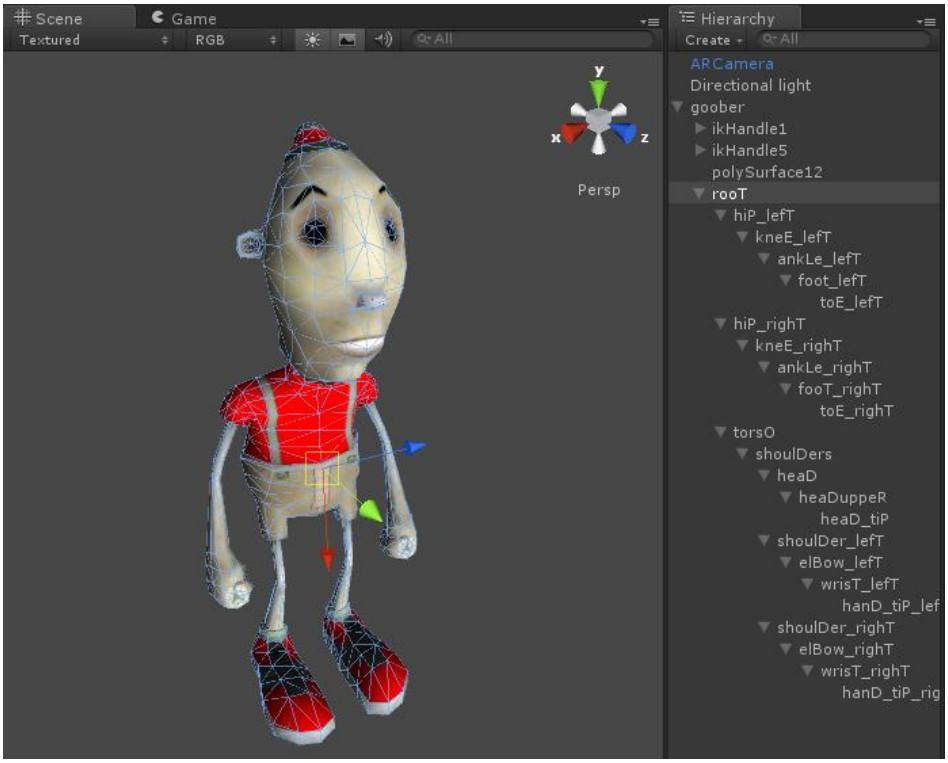


Slika 29 - Unity project

Na slici 29 mogu se vidjeti čitav Unity projekt, te pogledi koji pogledi *Hierarchy, Project, Scene i Inspector.*

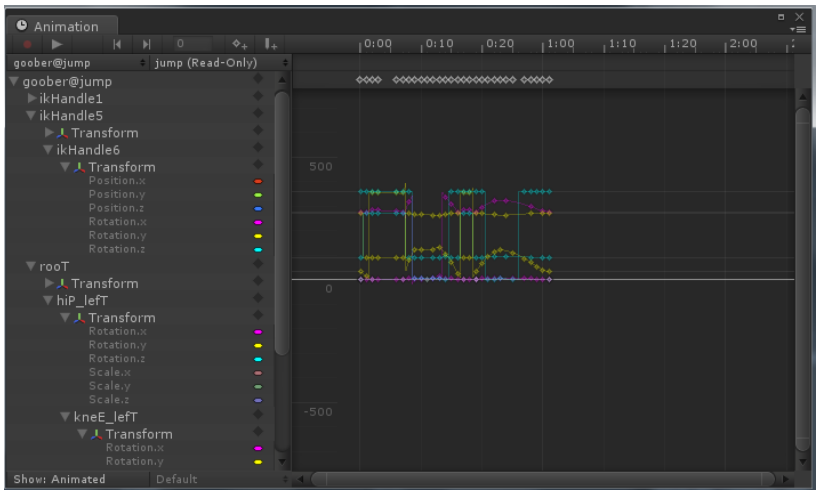
U *Hierarchy* pogledu moguće je vidjeti elemente scene: AR kameru, direkciono svijetlo, te ravnu površinu. Na tu površinu stavljen je *ImageTarget* marker, a na njega Virtualna tipka. Kao dijete Virtualnoj tipki stavljen je kinematički model – patuljak naziva *goober.* Pri postavljanju elemenata u odnos roditelj-dijete nasljeđuju se neke osobine komponenti scene, a omogućeno je i lakše upravljanje istim komponentama.

Radi se o kinematičkom modelu **FBX** formata. Na slici 30 prikazan je model, te hijerarhija istog. Model ima definirane tri animacije: *idle, jump i walk.*



Slika 30 – Hijerarhijski kinematički model

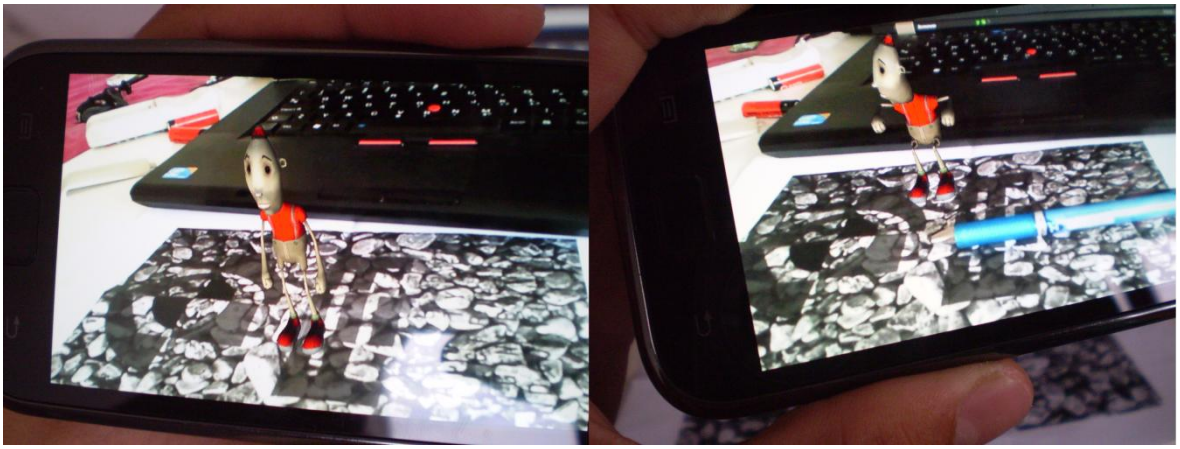
Bitno je spomenuti da Unity nudi alat za animiranje modela. Jednostavnim pomicanjem kostiju modela u određenim sekvencijama moguće je izraditi animaciju. Na slici 31 vidimo spomenuti alat.



Slika 31 - Unity Animation Window

Pri implementaciji Virtualnih tipki, potrebno je implementirati sučelje **IVirtualButtonEventHandler** dostupno u QCAR Extension paketu. Na osnovu implementacije ovog sučelja, prvenstveno metoda **OnButtonPressed i OnButtonReleased,** pozivaju se korespondentni događaji.

Na slici 32 prikazan je rad aplikacije.



Slika 32 - Interaktivni virtualni patuljak

Bitno je naglasiti da se Virtualna tipka inicira prekrivanjem detalja na markeru na poziciji tipke. Prema tome, zbog tehničkih detalja izvedbe samog slikanja mobitela, u ovom se primjeru koristi kemijska.

ZAKLJUČAK

Kroz ovaj diplomski rad proučavana je proširena stvarnost iz više perspektiva. Međutim, naglasak je bio na mogućnostima razvoja aplikacija temeljenih na proširenoj stvarnosti na Android mobilnoj platformi.

Prethodnih godina sam razvoj mobilnih uređaja doživljava eksponencijalni rast. Popratno tome, pojavila se Android mobilna platforma, Googleov *open-source* projekt. Rezultat kombinacije ove dvije činjenice je čitava vojska *developera* Android mobilnih aplikacija diljem svijeta.

Tako je u ovom diplomskog radu proučen jedan zanimljiv programski okvir (*framework)* – QCAR SDK koji omogućuje brži i jednostavniji razvoj aplikacija baziranih na kompleksnoj tehnologiji proširene stvarnosti. Kroz primjere, u radu je prikazan postupak razvoja takvih aplikacija. Ovim i ovakvim sustavima omogućen je razvoj širokog spektra aplikacija, te je samo kreativnost granica!

Nadalje, proučavani su sustavi proširene stvarnosti bez upotrebe markera, te je ostavljen prostor za daljnja izučavanja ove teme kroz kombiniranje ovakvih i ugrađenih sustava mobilnih uređaja.

LITERATURA I REFERENCE

[1] Ronald T. Azuma: *A Survey of Augmented Reality*, 1997.
http://www.cs.unc.edu/~azuma/ARpresence.pdf

[2] Paul Milgam, Haruo Takemura, Akira Utsumi i Fumio Kishino: *Augmented reality: A class of displays on the reality-virtuality continuum*, 1994.
http://www.cs.wpi.edu/~gogo/hive/papers/Milgram_Takemura_SPIE_1994.pdf

[3] Peter Antoniac: *Augmented Reality Based user interface for mobile applications and services*, 2005.
http://herkules.oulu.fi/isbn9514276965/isbn9514276965.pdf

[4] Feng Zhou, H.B.-L. Duh i M. Billinghurst. *Trends in augmented reality tracking,interaction and display*: *A review of ten years of ismar,* 2008.
http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4637362

[5] Katharina Pentenrieder, Christian Bade, Fabian Doil i Peter Meier: *Augmented reality-based factory planning - an application tailored to industrial needs*, 2007.
http://dx.doi.org/10.1109/ISMAR.2007.4538822

[6] Layar augmented reality browser
http://www.layar.com/

[7] Harris, C.: *Tracking with rigid models* 1992.

[8] Georg Klein: *Visual Tracking for Augmented Reality*
http://www.robots.ox.ac.uk/~gk/publications/Klein2006Thesis.pdf

[9] G. Klein, D. Murray: *Parallel Tracking and Mapping for Small AR Workspaces*
http://www.robots.ox.ac.uk/~gk/publications/KleinMurray2007ISMAR.pdf

[10] Arno Becker, Marcus Pant: *Android - Grundlagen und Programmierung. dpunkt.verlag*, 2009.

[11] Tobias Domhan: Augmented *Reality on Android Smartphones,* 2010
http://www.softwareforschung.de/fileadmin/_softwareforschung/downloads/WISTA/Tobias_Domhan_Studienarbeit.pdf

[12] Referenca na TooN biblioteku:
http://savannah.nongnu.org/projects/tooN

[13] Referenca na CVD i Gvars3 biblioteke:
http://savannah.nongnu.org/projects/libcvd

[14] Instalacijske instrukcije za PTAM sustav:
http://www.robots.ox.ac.uk/~gk/PTAM/README.txt

[15] Parallel Tracking and Multiple Mapping – stranica projekta
http://www.robots.ox.ac.uk/~bob/research/research_ptamm.html

[16] AndAR - Android Augmented Reality – stranica projekta
http://code.google.com/p/andar/

[17] QCAR SDK – stranica projekta
https://ar.qualcomm.com/qdevnet/sdk

[18] Skripta za generiranje grafičkih modela u .h *header* formatu:
http://heikobehrens.net/2009/08/27/obj2opengl/

[19] Unity3D – stranica projekta:
http://www.unity3d.com/

[20] Zakrpa za ignoriranje iznimaka pri pokretanju programa na Windows x64
http://support.microsoft.com/kb/976038

[21] http://www.robots.ox.ac.uk/~bob/research/research_ptamm.html

SAŽETAK

Glavna tema ovog diplomskog rada je proširena stvarnost, te mogućnosti prikaza kinematičkog modela na ugrađenom sustavu. Obuhvaćena je temeljna teorija proširene stvarnosti, razvoj kroz godine, te osnovna područja primjene.

U drugom poglavlju proučena je teorija i jedna implementacija sustava proširene stvarnosti bez upotrebe markera. Proučen je sustav PTAM – „Parallel Tracking and Maping“.

U trećem je poglavlju proučen ugrađeni sustav na temelju Android platforme, te temeljne značajke bitne za daljnje implementacije sustava na navedenoj platformi.

U četvrtom poglavlju inkorporirani su segmenti proširene stvarnosti na ugrađenom sustavu – Android platformi. Na temelju QCAR SDK prikazan je razvoj aplikacija temeljenih na ovim načelima na primjeru jednostavnih aplikacija. Omogućen je prikaz modela u realnoj okolini putem kamere, te interakcija s modelima putem virtualnih tipki.

ABSTRACT

The main theme of this thesis was augmented reality, and displaying kinematic models on integrated systems. It covers the basic theory of augmented reality, developing through the years and the main areas of application.

The Second chapter examined the theory and implementation of an augmented reality system without using markers. We have analyzed the system PTAM – “Parallel Tracking and mapping”.

The third chapter studied an embedded system based on the Android platform, and the basic features essential for the further implementation of the platform.

In the fourth chapter are incorporated segments of augmented reality for embedded systems. Based on QCAR SDK, simple Android applications are developed, showing the models in a real environment through a camera, and interacting with the models by virtual keys.