Sustav za komunikaciju u stvarnom vremenu korištenjem tehnologije miješane stvarnosti

Projektna dokumentacija

Verzija 1.0

Studentski tim: Oleg Jakovljev

Renato-Zaneto Lukež

**Nastavnik:** prof. dr. sc. Željka Mihajlović

Tehnička dokumentacija

# Tijek razvoja sustava

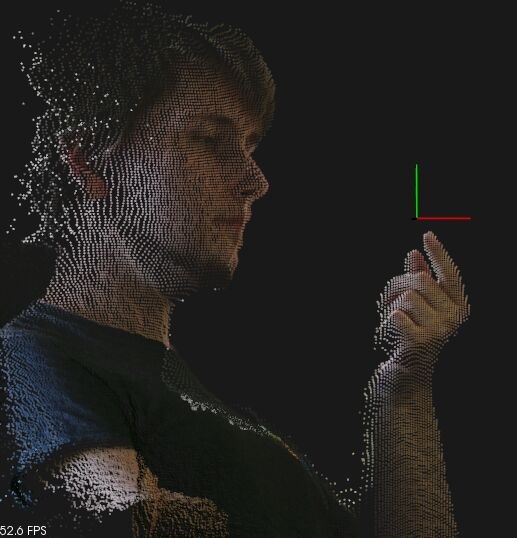
Tijek razvoja cjelokupnog sustava je veoma bitno opisati zbog činjenica da su brojna rješenja odbačena zbog raznih razloga, među kojima su najčešći bili zahtjev za brzinom rada sustava (sustav za komunikaciju u stvarnom vremenu; brzina odziva i prikaza je ključna za prirodno korištenje programskog rješenja) te mogućnost implementacije na hardveru trenutne generacije (latencije pri obradi te slanju podataka, primarno). Ponajviše problema izaziva ograničenje propusnosti pri prijenosu preko mreže. Potrebno je prenijeti objekt (engl. *mesh*) korisnika na poslužitelju kojega stvara dubinska kamera. Prva je ideja klijentu slati sve točke objekta i teksturu objekta za rekonstrukciju [4].

Prilikom istraživanja radova na temu prijenosa točaka objekta naišli smo na rad koji govori o prijenosu visoko kvalitetnog video toka slobodnog gledišta [8] u kojem je predstavljeno rješenje za prijenos takvog videa kao kompaktnog toka podataka. U radu se također spominje korištenje gustog sklopa RGB i infracrvenih videokamera koje stvaraju dinamički teksturiranu površinu koja se zatim kompresira u prenosivi 3D video format. U cilju pokušaja rekonstrukcije takvog sustava istražili smo mogućnost korištenja metoda iz nekoliko sličnih radova.

Ako govorimo o rekonstrukciji modela iz objekta točaka u mrežasti model do sada najbolje opisana metoda, a ujedno i metoda koja se koristi u svim navedenim radovima, je metoda Poissonove rekonstrukcije površine. [13] Glavni razlog korištenje ove metode je zbog njene smanjene vremenske kompleksnosti gdje se rješenje svodi na linearan broj točaka, te se time omogućuje brža rekonstrukcija i veća kvaliteta rekonstruirane površine.

Ako govorimo o prikazu objekta točaka, jedan od takvih radova [9] omogućuje snimanje zahtjevnih scena i izgradnju novih modela iz postojećih u stvarnom vremenu (odvajanje spojenog modela lopte i čovjeka u dva zasebna modela). Iako ima zanimljiv rezultat, ovaj rad ne govori puno o mogućnosti prijenosa modela putem mreže, nego isključivo govori o izgradnji i prikazu složenih modela u stvarnom vremenu. Isto tako proučili smo rad [10] o rekonstrukciji ne krutih volumetrijskih modela u stvarnom vremenu. Problem kod ovog postupka je prvo to što mu treba vremena za skeniranje modela da bi se taj model uopće izgradio, pa obrada i stvaranje modela. Isto tako modeli koji se izgrade su statični, dok naš postupak zahtjeva skeniranje dinamičnog pokretnog modela. Drugi članak na istu temu poput [11] također govori o skeniranju modela, ali također isključivo za statički model.

U konačnici naišli smo na rad [12] koji govori o kodeku koji omogućuje prijenos generičkog objekta točaka u stvarnom vremenu za 3D video. Ovaj kodek prikladan je za aplikacije miješane stvarnosti gdje se 3D objekt točaka hvata brzom stopom. Osim što omogućuje primjenu prediktivnog algoritma za predikciju pojedinih blokova u prediktivnom okviru, omogućuje i kompresiju samog skupa točaka. Za prijenos velike količine podataka u stvarnom vremenu tradicionalno se koristi UDP protokol no količina podataka potrebna za prijenos objekta točaka u stvarnom vremenu puno je veća od propusnosti prosječnih mreža dostupnih korisnicima, te to vodi do potrebe za kompresijom podataka što nam ovaj kodek upravo omogućuje. Uz to ovaj kodek očekuje odobrenje MPEG standarda za prikaza 3D skupa točaka u stvarnom vremenu. Upravo iz tih razloga odlučili smo dublje istražiti ovaj rad. Metoda opisana u radu [12] zahtjeva korištenje PCL biblioteke (engl. *point cloud library*) koja pruža niz metoda za upravljanje objektom točaka. Nakon istraživanja metoda kompresije i primjenom kodeka dobiva se smanjena veličina podataka (za cca 10x kompresijom). Osim kompresije podataka, moguće je smanjiti količinu podataka za prijenos filtriranjem korisnika iz ukupne dubinske slike rezanjem pozadine. Tako još dodatno smanjimo veličinu podataka. Kompresija ne uništava oblak točaka (engl. *point cloud*), nego ga razrijedi i kao takav se direktno može koristiti za prikaz u stvarnom vremenu. Rezultat kompresije ovim kodekom prikazuje Slika 1.

****

**Slika 1.** - Kompresirani objekt točaka

Istraživanjem drugih opcija razmotrena je mogućnost prenošenja dubinske slike te rekonstrukcije samog objekta na strani klijenta. Prijenos dubinske slike iziskuje više izračuna na klijentskoj strani no omogućuje prijenos značajno manje podataka. Razmotrimo sljedeće: dubinska kamera uređaja Microsoft Kinect V2 ima razlučivost 512 x 424 točaka. Za opis dubine na pojedinoj točki slike potrebno je 11 bitova. Uz aproksimaciju preciznosti tj. primjenu ravnina odsijecanja to je moguće smanjiti na 8 bitova po točki. U suprotnosti, za prijenos punog oblaka točaka (engl. *point cloud*) potrebne su x, y i z koordinate za svaku točku u prostoru, što daje 3\*32 bita = 96 bitova po točki. Prednosti prijenosa dubinskog okvira slike su očite zbog višestrukoga smanjenja veličine pojedine operacije prijenosa. Ova, druga ideja, ima i prednost time da ako se pojedini okvir dubine (engl. *depth frame*) predstavi kao slika sivih razina (engl. *grayscale*), moguće je nad njom primijeniti postojeće opće podržane metode za kompresiju i tok slika (engl. *streaming*). Kompresija oblaka točaka također je i teže ostvariva zbog nekompatibilnosti između postojećih algoritama za kompresiju oblaka točaka i podatkovnih struktura te programskih sučelja koja koristi Microsoft Kinect V2, kojega koristimo za stvaranje dubinske slike korisnika. Od ostalih ideja, postoji mogućnost skeniranja korisnika prije pojedinog razgovora te slanje modela klijentima jednom. U tom bi se slučaju u stvarnom vremenu prenosio sam položaj korisnika i orijentacije ključnih točaka na njegovom tijelu prilikom pomicanja kroz korištenje sustava. Razmotrivši sve navedeno, odlučili smo s poslužitelja slati dubinski okvir s Kinect senzora a rekonstrukciju objekta delegirati klijentskom uređaju – Microsoft Hololens raspolaže popriličnom procesnom moći u obliku dediciranog grafičkog procesora pa je stoga moguće i na njemu obavljati rekonstrukciju objekta iz dubinskog okvira. Nakon ove odluke uslijedilo je odlučivanje o ustroju sustava i oblikovanju njegovih dijelova tako da se olakša razvoj, ispitivanje i omogući dinamika pri implementaciji novih mogućnosti.

# Opis razvijenog proizvoda

Projekt se temelji na istraživanju i razvoju podloge za komunikacijsku platformu u miješanoj stvarnosti pomoću 3D vizualizacije korisnika. Kroz rad na projektu istraženi su brojni načini rješavanja pojedinih problema koji se javljaju pri prikazu, obradi i slanju podataka preko mreže u stvarnom vremenu. Neke su metode obrađene površno a one koje su se pokazale najboljima su odabrane za implementaciju pomoću grafičkog pogona Unity3D.

## Implementacija programskog rješenja u Unity3D

Radni okvir (engl. *framework*) za razvoj ovoga sustava smješten je u Unity3D u niz neovisnih, izmjenjivih cjelina koje omogućavaju fleksibilno i agilno testiranje te razvoj pojedinih komponenti sustava. Pošto je sustav namijenjen komunikaciji putem mreže dvije su glavne cjeline poslužitelj i klijent (koncept proširiv na više klijenata). Inicijalno, sustav je namijenjen emitiranju (engl. *broadcastu*)podataka s poslužitelja prema klijentima a na strani poslužitelja bi trebao biti KinectV2 uređaj koji služi za stvaranje mreže poligona (engl. *mesha*) korisnika koji se prenosi mrežom do klijenata. Klijentska komponenta se bavi prikazom mreže poligona korisnika primljenog s poslužitelja putem naočala za miješanu stvarnost (trenutno Microsoft Hololens). Serverska i klijentska cjelina svake imaju po tri komponente.

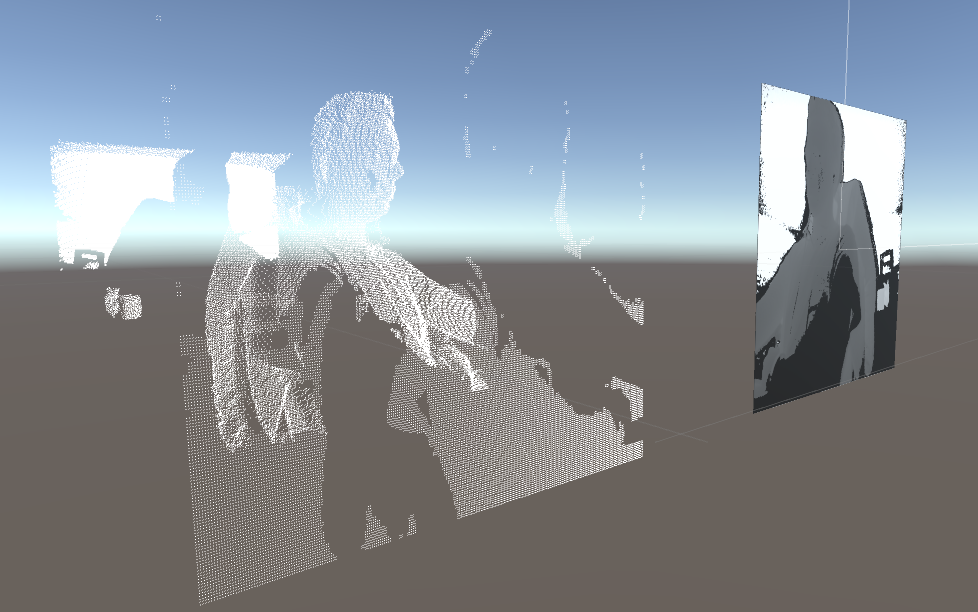
Serverska komponenta redom zove svoje tri komponente: učitavanje podataka s nekog izvora (KinectV2, tvrdi disk, video zapis, itd.), obrada podataka (kompresija, izrezivanje korisnika, optimizacije, itd.) te slanje podataka (putem Unity Uneta, TCP ili UDP veze). Moguće je odvojeno testiranje poslužiteljske komponente posebno od klijentske te testiranje pojedinih cjelina unutar nje jednostavnim izmjenjivanjem komponenti koje komuniciraju putem dogovorenog sučelja.

Slična izvedba koristi se za klijentsku cjelinu koja, slično, zove svoje tri analogne komponente: primanje podataka s nekog izvora (Unity Unet, TCP/UDP, tvrdi disk, videozapis, itd.), obrada podataka (eventualna dekompresija, rekonstrukcija itd.) i vizualizacija podataka (ovisno o podacima; *point cloud*, vizualizacija mesha, prikaz na 2D teksturi, itd.). Ovakva je izvedba odabrana zbog jednostavne izmjenjivosti komponenti te mogućnosti razvoja komponenti neovisno, a prednost koja se dobiva mogućnošću korištenja izvornih programskih dodataka (engl. *native* plugina) u Unity3D daje sustavu sposobnost da poziva i vanjske komponente pisane u drugim jezicima (C/C++) te na drugim platformama. Kako je težnja trenutne faze projekta istraživanje tehnoloških mogućnosti, kroz nekoliko mjeseci rada na projektu istražena su brojna rješenja koja se odnose na većinu navedenih komponenti sustava. Slika 2. prikazuje oblikovanje komponenti sustava.

Untitled Diagram

Slika 2. - Dijagram sustava

Slika 3. prikazuje konačan rezultat razvoja sustava u trenutačnoj iteraciji. Zbog izvođenja na istom računalu prilikom razvoja, sve se odvija u jednoj Unity3D sceni. Na lijevoj je strani prikaz dubinskog okvira Kinect senzora kao teksture na serverskoj cjelini. Ovdje se vidi redundancija u prikazu 11 bitnih podataka kao RGBA32 tekstura koji se vizualiziraju kao sivih razinaslika s 11 bitnom dubinom. Na lijevoj je strani prikaz rekonstruirane reprezentacije korisnika prikazane metodom prikaza oblaka točaka.



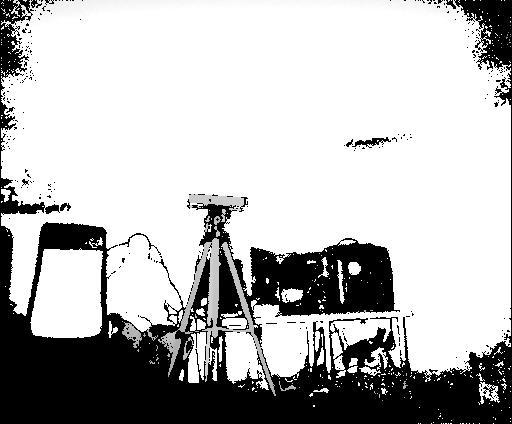
Slika 2. – Prikaz rada klijentske i poslužiteljske cjeline u istoj sceni

# Poslužiteljska cjelina sustava

Za dogovaranje sučelja komponenti na poslužitelju zaslužna su tri sučelja od kojih svako definira jednu funkciju: sučelje**DataSource** s metodom **Texture2D** **fetchIt( void )**, sučelje **DataProcessor** s metodom **byte[] processIt( Texture2D )** te sučelje **DataSender** s metodom **void sendIt ( byte[] )**. Metode se pozivaju iz skripte koja ih poziva jednu za drugom bez dodatne obrade podataka.

## Komponenta učitavanja podataka

U ranim fazama razvoja sustava te testiranja ostalih komponenti radi jednostavnosti koristili smo učitavanje statične slike s tvrdog diska. Naknadno je napisano sučelje za dohvat podataka iz Microsoft Kinect V2 uređaja pomoću njegovog programskog sučelja. Ono prosljeđuje dubinski okvir iz senzora, pretvara ga u teksturu te vraća tu teksturu. Tekstura se trenutno, radi jednostavnosti, sastoji od četiri iste RGBA komponente koje predstavljaju sliku sivih razina. Jednostavna optimizacija svodi se na prijelaz s formata slike *rgba32* na format sivih slika. U ovoj komponenti dolazi do najvećeg gubitka performansi zbog skupe operacije učestalog stvaranja teksture iz dubinskog okvira – naime rad s teksturama uvjetuje pristup grafičkoj memoriji zbog načina izvedbe grafičkog pogona Unity3D, a učestalo stvaranje teksture u memoriji (oko 30 puta u sekundi) povlači i učestalo kopiranje podataka iz radne memorije (dubinski okvir) u radnu memoriju grafičke kartice (tekstura). Srećom, ovaj se posao može izbjeći u fazama projekta kad vizualizacija u svrhu ispitivanja i provjere nije više potrebna. Slika prikazuje dubinski okvir uređaja KinectV2 preciznosti 11 bita po točki prikazan kao RGB slika koji se obrađuje i prenosi klijentu.



Slika 3. – Dubinski okvir pretvoren u sliku sivih razina

## Komponenta obrade podataka

Komponenta za obradu podataka iziskuje najviše istraživanje i ključna je za rad sustava u stvarnom vremenu. Pri implementaciji obrade podataka treba paziti na brzinu obrade koja mora biti iznimno niska i efikasnost kompresije da bi se omogućio prijenos dubinskog okvira preko mreže. Glavni smjer istraživanja je korištenje programa FFmpeg za enkodiranje toka podataka (engl. *streama*) s dubinskog senzora u mpeg4 tok podatakakoji je pogodan za prijenos mrežom. Kako je FFmpeg loše dokumentiran i nema puno primjera korištenja, većina programera se odlučuje na pozivanje FFmpeg izvršnog programa iz vlastitog programa. Izvršni program distribuira se za više operacijskih sustava a nudi većinu mogućnosti koje nudi i programsko sučelje. Za enkodiranje i dekodiranje u mpeg4 pomoću x264 kodeka postoji i sklopovska podrška na većini učestalih procesora, konkretno Intel Quick Sync [3], Nvidia NVEnc i ostali, no upitno je mogu li prijenosni uređaji poput Microsoft Hololensa koristiti neke od tih metoda. Pri usporedbi različitih sklopovskih i softverskih rješenja utvrđeno je [7] da najbolje rezultate daju upravo Intel Quick Sync i NVEnc (no u slučaju korištenja NVEnca postoji poprilično kašnjenje zbog kopiranja podataka iz radne memorije računala u grafičku memoriju). Pristup mogućnostima FFmpeg APIja moguć je na nekoliko načina [1]. Za najviše kontrole moguće je skinuti potpuni izvorni kod te ga izgraditi u vlastitom projektu. Tako je moguće pristupiti najnižim slojevima i podacima s kojima radi programsko sučelje FFmpega. Drugi je način korištenje gotovih prethodno izgrađenih biblioteka koje nude mnoge funkcije i strukture podataka za korištenje u vanjskim projektima kroz pristupanje dinamičkim bibliotekama. Treći je način korištenje izvršne verzije programa koja je izgrađena za pojedini operacijski sustav. Ovo nudi najmanje fleksibilan pristup kako se sve mora raditi preko argumenata komandne linije koji imaju određena ograničenja. Ipak, zbog zahtjeva za brzinom testiranja i brzog upoznavanja mogućnosti sustava, odabran je treći način, a zbog visoke modularnosti sustava u kasnijim će se fazama primijeniti specijalizirane izgrađene biblioteke radi maksimalne optimizacije performansi. Uz prijenos teksture dubinskog okvira Kinecta potrebno je prenositi i teksturu korisnika čiji model stvaramo i prenosimo klijentima. Za to će se koristiti običan mp4 *stream*[6]koji je uobičajen u primjenama video prijenosa u stvarnom vremenu. Prijenos podataka iz Unity3D u FFmpeg je također problematičan. Potrebno je koristiti cjevovode [5] za prijenos podataka u te iz izvršnog programa FFmpeg. Parametri se u izvršni program zadaju argumentima komadne linije te tako zadajemo izvor i odredište podataka, format te opis formata, veličinu ulazne slike, korištene kodeke itd. Za prijenos podataka koristimo imenovane cjevovode koje podržava operacijski sustav Windows no javljaju se brojne poteškoće pri otkrivanju pogreški u slučajevima kad FFmpeg ne uspije dati izlaz i opis izračuna.

Trenutno je za kompresiju implementirana klasa **DeflateStream**koja je implementiram u C# biblioteci System.IO.Compression. Ona svoju namjenu vrši sasvim adekvatno zbog velike redundancije u podacima dubinskog okvira RGBA32 formata, smanjujući veličinu pojedinog okvira s 868352 byteova na oko 30 do 50 kilobajta prije prijenosa mrežom. Brzina kompresije je također zadovoljavajuća za trenutačnu iteraciju razvoja sustava, kao i kvaliteta rekonstruirane slike.

## Komponenta slanja podataka klijentu

Slanje podataka inicijalno je ostvareno korištenjem Unity3D transportnog sloja koji je zamišljen za jednostavan prijenos podataka preko mreže u raznim aplikacijama i igrama. Ubrzo se ipak pokazalo da je taj sustav neadekvatan za postavljene zahtjeve te podbacuje po pitanju propusnosti, brzine i pouzdanosti, ovisno o konfiguraciji korištenja. Zato je razvijena komponenta koja koristi brzu i pouzdanu TCP vezu za slanje potrebnih podataka. Pri testiranju u lokalnoj je mreži uspješno ostvarena komunikacija s propusnošću od 90 megabita po sekundi, ili oko 12 megabajta u sekundu. Za prijenos podataka koriste se klasa **NetworkStream**koja je implementiran u bibliotekama jezika C# koju koristimo za razvoj skripti u Unity3D.

# Klijentska cjelina sustava

Za dogovaranje sučelja komponenti na klijentu postoje tri sučelja od kojih svako definira jednu funkciju: sučelje**ClientReceive** s metodom **byte[]** **receiveIt( void )**, sučelje **ClientProcess** s metodom **Texture2D processIt( byte[] )** te sučelje **ClientVisualize** s metodom **void showIt (Texture2D)**. Metode se pozivaju iz skripte koja ih poziva jednu za drugom bez dodatne obrade podataka, analogno načinu rada poslužitelja.

## Komponenta primanja podataka

Primanje podataka mora odgovarati načinu slanja pa je tako razvijena komponenta koja koristi Unity3D sučelje za mrežno programiranje i TCP veza koja prima podatke s poslužitelja i prosljeđuje ih idućoj komponenti. Vrijedi napomenuti da i FFmpeg nudi mogućnost izravnog slanja preko specificiranog protokola, na specificiranu mrežnu adresu, no u trenutnoj iteraciji projekta te mogućnosti nisu istražene.

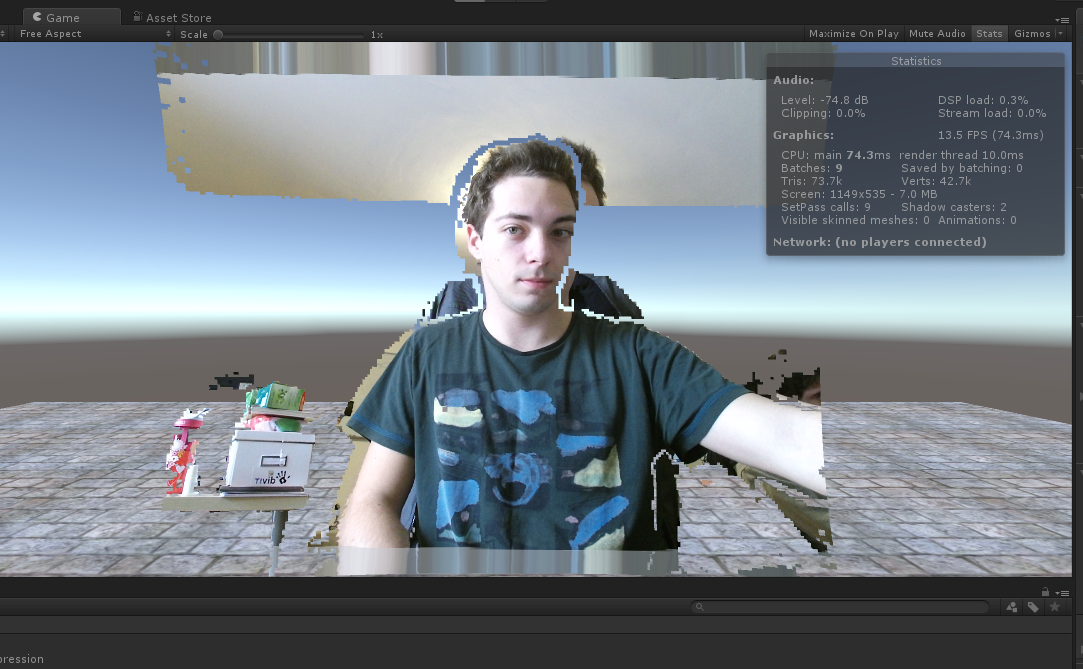
## Komponenta obrade podataka

Komponenta obrade podataka mora biti analogna onoj korištenoj kod klijenta. U ovome se slučaju koristi **DeflateStream**za dekompresiju no performanse koje je moguće ostvariti pomoću prijenosa podataka enkodiranih u mpeg4 format su mnogo bolje od trenutne implementacije. Kompresija pomoću **DeflateStreama**trenutno funkcionira iznimno dobro zbog velike redundancije podataka na poslužitelju no s teksturom boje s RGB kamere KinectV2 uređaja priča je potpuno drugačija jer se radi o slici mnogo veće razlučivosti (1920x1080 točaka) koja se neće moći niti približno dobro komprimirati ovim metodama. Ovdje uskače mpeg4 tok podatakakoji je predviđen za upravo tu primjenu prijenosa slike u stvarnom vremenu i omogućava iznimno dobre performanse čak i pri kompresiji bez gubitaka.

## Komponenta vizualizacije podataka

Početno, pri testiranju ispravnosti rada prenošenja dubinskog okvira korištena je vizualizacija dubinskog okvira u obliku RGB slike na teksturi. Cilj je bio pojednostaviti prikaz i proces približne procjene kvalitete prenesenog okvira. Kako bi se prikazala na teksturi korištenjem Unityeve **Renderer**komponente, primljena tekstura se prvo mora deserijalizirati iz podatkovnih okvira u kojima je prenesena, u obliku polja bajtova.

U finalnoj fazi projekta taj je način unaprijeđen u prikazu oblaka točak**a**, koristeći komponentu za prikaz koja iz polja bajtova primljenog preko mreže stvara podatke za trodimenzionalni prikaz točaka u prostoru. Pritom se koristi približna pretvorba iz vrijednosti zapisanih na pojedinoj koordinati teksture u prostornu poziciju za točku prikaza. Važno je napomenuti da je i ovaj način prikaza samo privremena stepenica u iterativnom razvoju sustava prema konačnom proizvodu. U nastavku slijedi usporedba načina prikaza oblaka točaka rekonstrukcijom mreže poligonai izravnim iscrtavanjem točaka. Na prvoj slici vidimo objekt rekonstruiran pomoću poligona s dodanom teksturom.



Slika 4. – Vizualizacija oblaka točaka metodom poligonizacije

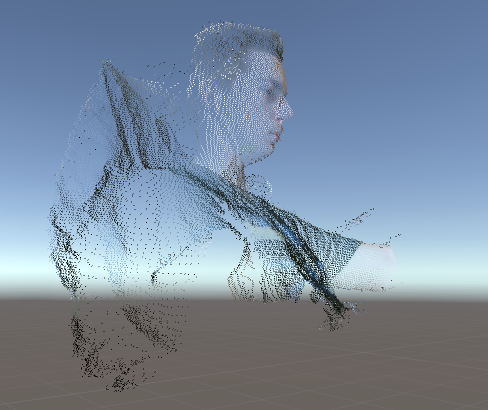
Druga slika prikazuje izgled oblaka točaka prikazanog bez poligonizacije, pobojanog s teksturom. Vidi se da je prvi način prirodniji i vjernije prikazuje osobu u prostoru, no trenutno se koristi drugi način jer je prikladniji za svrhu testiranja i procjene utjecaja kompresije i gubitka podataka na rezultat.



Slika 5. – Vizualizacija oblaka točaka sirovim prikazom

Oba načina se razlikuju samo u krajnjoj fazi pripreme objekta za prikaz i podržavaju ravnine odsijecanja (po dubini i širini). Metode su slične onima korištenim u KinectV2 primjerima za Unity3D i oslanjaju se na podatke iz Kinect Managera. Kako on nije dostupan na klijentskoj strani, uspješno je emuliran prikladnim objektom omotačem koji omogućava pristup polju **depth2SpaceCoords**koji u sebi drži vrhove za prikaz oblaka točaka u prostoru.

Pri pogledu na rekonstruiranu reprezentaciju korisnika sa strane (ili iz kuta iz kojeg Kinect senzor ne snima korisnika) vide se greške u uzorkovanju koje su uzrokovane korištenjem samo jednog Kinect senzora. Kako bi se ovo izbjeglo u konačnoj verziji proizvoda, predviđeno je uzorkovanje korisnika uz pomoć više senzora postavljenih oko korisnika tako da ga mogu u cijelosti skenirati. Slika prikazuje oblak točaka iz Kinect senzora, pogledan sa strane te iz blizine.



Slika 6. – Vizualizacija oblaka točaka gledana s bočne strane

Sustav u trenutnom stanju tako omogućava prikaz očitane mreže poligonaiz KinectV2 uređaja na udaljenom klijentu koji je dostupan preko mreže. Pri testiranju u lokalnoj mreži utvrđeno je da se propusnost potrebna za prijenos ovih podataka kreće između 3 i 4 megabita po sekundi. Prednost ovog sustava je što se većina postojeće infrastrukture sustava može ponovno iskoristiti bez ikakvih promjena za komuniciranje same teksture boje s poslužitelja na klijentsku stranu.

# Planirana poboljšanja

U trenutnoj fazi razvoja teško je predvidjeti koje će se tehnologije koristiti u konačnoj verziji proizvoda no idući su koraci jasni. Ključno je uvesti i prijenos RGB teksture u posebnom toku podatakapomoću enkodiranja u mpeg4 format. Zamjenom kompresije ispuhavanja (engl. *deflate compression)* s navedenom metodom rezultirat će smanjenjem potrebne propusnosti, a ovisno o mogućnosti korištenja *Intel Quick Sync* sklopovske podrške i poboljšanjem u performansama kompresije. Veće će se optimizacije uvesti i kvalitetnim izrezivanjem korisnika od pozadine te slanjem samo bitnih podataka u dubinskom okviru. Nije još jasno koliki su sklopovski zahtjevi enkodiranja i hoće li neovisni Microsoft Hololens uređaj moći izvršavati dekodiranje u stvarnom vremenu. Nadalje, treba zamijeniti prikaz na klijentu u Unity3D s prikazom na Hololens uređaju (Unity3D ima podršku za rad s Hololens naočalama za miješanu stvarnost) i testirati performanse. Potrebno je i optimizirati memorijske i procesorske zahtjeve poslužiteljske i klijentske obrade kroz softverske prečace i korištenje naprednijih hardverskih tehnika poput višedretvenosti i sklopovske kompresije tj. enkodiranja. Ovisno o mogućnostima dostupnih tehnologija treba naći ravnotežu između kvalitete i brzine pojedinih komponenti. Jedno od poboljšanja sustava je uvođenje više dubinskih kamera koje korisnika snimaju s više strana i pouzdanih algoritama za spajanje oblaka točaka i poligonizacije istih. Na kraju, treba isprobati i sustav u okruženjima dvosmjerne komunikacije, koja je jedan od ciljeva razvoja ovoga sustava.

# Literatura

[1] **FFmpeg documentation.** *FFmpeg Web site.* [Mrežno] FFmpeg.org. [Pristupano: Prosinac, 2016.],

<https://www.ffmpeg.org/ffmpeg.html>

[2] Communications & Multimedia Laboratory, University of Taiwan **. Coding of audio-visual objects – MPEG4 Part 14.** [Mrežno], 15. Studeni 2003, http://www.cmlab.csie.ntu.edu.tw/~cathyp/eBooks/ 14496\_MPEG4/ISO\_IEC\_14496-14\_2003-11-15.pdf

[3] Intel, **Quick Sync video hardware acceleration** [Mrežno], <https://software.intel.com/en-us/articles/using-the-intel-media-sdk-within-mainconcept-h264avc-encoder-for-intel-quick-sync-video>

[4] Microsoft, **Lossless depth frame coding** [Mrežno], Veljača 2016, <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/depthcode-final.pdf>

[5] Microsoft, *Microsoft Developer Network*, **Pipelines** [Mrežno], <https://msdn.microsoft.com/en-us/library/ff963548.aspx>

[6] Dapeng Oliver Wu, *University of Florida,* **FFmpeg real time encoding/decoding, x264 codec** [Mrežno], [http://www.wu.ece.ufl.edu/projects/wirelessVideo/project/realTimeCoding/ download/doc/howto.pdf](http://www.wu.ece.ufl.edu/projects/wirelessVideo/project/realTimeCoding/download/doc/howto.pdf)

[7] OBS Forum, *Open Broadcaster Software*, **OBS benchmarking: CPU vs NVENC vs Quick Sync** [Mrežno], Srpanj 2014, [Pristupano: Siječanj, 2017.], [https://obsproject.com/forum/ threads/obs-benchmarking-1080p-60fps-cpu-vs-nvenc-vs-quick-sync.15963/](https://obsproject.com/forum/threads/obs-benchmarking-1080p-60fps-cpu-vs-nvenc-vs-quick-sync.15963/)

[8] Microsoft, **High-Quality Streamable Free-Viewpoint Video**, http://research.microsoft.com/en-us/um/redmond/projects/fvv/, 2015

[9] Microsoft, **Fusion4D: Real-time- performance capture of challenging scenes**, http://perceptiveio .com/publications/2016/7/13/fusion4d-real-time-performance-capture-of-challenging-scenes, Srpanj 2016.

[10] [Matthias Innman](http://lgdv.cs.fau.de/people/card/matthias/innmann/), [Michael Zollhöfer](http://people.mpi-inf.mpg.de/~mzollhoef/), [Matthias Nießner](http://graphics.stanford.edu/~niessner/publications.html), [Christian Theobalt](http://people.mpi-inf.mpg.de/~theobalt/), [Marc Stamminger](http://lgdv.cs.fau.de/people/card/marc/stamminger/), *University of Erlangen-Nuremberg, Max Planck Institute for Informatics, Stanford University*, **VolumeDeform: Real-time Volumetric Non-rigid Reconstruction**, http://lgdv.cs.fau.de/ publications/publication/Pub.2016.tech.IMMD.IMMD9.volume\_6/, 2016

[11] [Matthias Innman](http://lgdv.cs.fau.de/people/card/matthias/innmann/), [Michael Zollhöfer](http://people.mpi-inf.mpg.de/~mzollhoef/), [Sharami](http://graphics.stanford.edu/~niessner/publications.html) Izadi, [Marc Stamminger](http://lgdv.cs.fau.de/people/card/marc/stamminger/), *University of Erlangen-Nuremberg, Microsoft Research Cambridge, Stanford University*, **Real-time 3D Reconstruction at Scale using Voxel Hashing**, http://cdn.octo-dev.co.uk/publications/niessner2013hashing.pdf, 2013

[12] R. Merkuria, K. Blom, P.Cesar, **Design, Implementation and Evaluation of a Point Cloud Codec for Tele-Imersive Video**, http://l.web.umkc.edu/lizhu/teaching/2016sp.video-communication/ref/Mekuria2015.pdf, 2015

[13] Michael Kazhdan, Hugues Hoppe, **Screened Poisson Surface Reconstruction**, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.302.2091&rep=rep1&type=pdf, 2012

Proceduralno generiranje gradova na temelju OpenStreetMap podataka

Projektna dokumentacija

Verzija 1.0

Studentski tim: Luka Kunić

Nastavnik: prof. dr. sc. Željka Mihajlović

Tehnička dokumentacija

# Opis razvijenog proizvoda

OpenStreetMap (OSM) je servis otvorenog koda koji omogućuje slobodno dohvaćanje geografskih podataka. Podaci se dohvaćaju u obliku XML datoteke koja sadrži elemente karte koja se crta. Svi elementi karte grade se od čvorova (engl. *nodes*) koji su definirani koordinatama i jedinstvenim identifikacijskim brojem. Čvorovi se u elemente povezuju putevima (engl. *ways*) koji sadrže informacije o vrsti elementa, popis svih čvorova od kojih se sastoji taj element i dodatne informacije koje ovise o vrsti elementa. Vrste elemenata uglavnom su ceste i građevine, ali elementi mogu biti i razne površine sa specifičnom namjenom, kao što su polja, parkirališta, parkovi ili vodene površine. Ovakav način pohrane podataka korisniku daje slobodu da na različite načine prikaže preuzetu kartu, kao i izbor elemenata koji će u konačnici biti prikazani.

Ideja ovog projekta bila je generirati 3D model grada iz proizvoljno odabranih OSM podataka. Zbog jednostavnosti, u obzir su uzeti samo elementi koji opisuju ceste i zgrade. Zgrade su u OSM datoteci definirane tlocrtnim obrisima, a ceste su definirane lancima čvorova.

Za generiranje modela korišten je Blender i njegov Python API koji omogućuje programsko upravljanje aplikacijom i alatima za izradu modela. Izrađen je dodatak (engl. *add-on*) za Blender koji dodaje alat za učitavanje .OSM datoteka.

# Tehničke značajke

## Izgled .OSM datoteke

Svaka .OSM datoteka sastoji se od četiri korijenska čvora: *bounds*, *node*, *way* i *relation*:

* *Bounds –* Jedan čvor na početku datoteke koji sadrži rubne koordinate svih čvorova u datoteci. Podaci iz ovog čvora koriste se za preračunavanje koordinata svakog čvora iz geografskog sustava u kartezijev kako bi bili ispravno prikazani u 3D prostoru.
* *Node –* Sadrži informaciju o jednom OSM čvoru: jedinstveni identifikator, geografske koordinate čvora i može sadržavati dodatne podatke koji ovise o vrsti čvora (npr. prometne oznake, stanice javnog prijevoza, imena ulica i sl.)
* *Way –* Povezuje čvorove u puteve. Svi čvorovi koji su dio nekog puta moraju biti ranije navedeni kao *Node*-elementi. Sadrži informaciju o vrsti elementa koju put predstavlja (npr. zgrada, cesta, zelena ili vodena površina, itd.)
* *Relation –* Služi za povezivanje puteva i čvorova u cjeline. Primjerice, relacijama se mogu definirati ulice koje se sastoje od više povezanih puteva, ili primjerice linije javnog prijevoza.

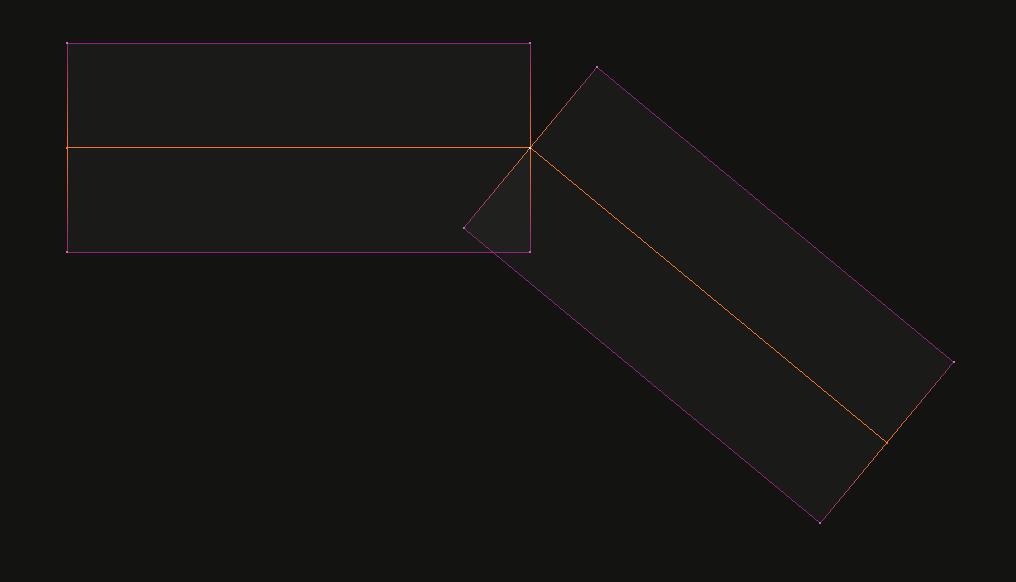
U sklopu ovog projekta korišteni su samo podaci o čvorovima i putevima. Relacije nisu relevantne za generiranje modela jer služe samo kao dodatna informacija o tome kako su čvorovi i putevi međusobno povezani.

## Strukture podataka

Pročitani podaci spremaju se u listu čvorova i listu puteva. Svaki put sadrži listu čvorova od kojih se sastoji, a svaki čvor sadrži listu puteva čiji je on dio. Jedan čvor može biti dio više povezanih puteva, primjerice ako se radi o križanjima. Dodatne informacije o čvorovima i putevima pohranjene su u obliku ključ-vrijednost parova.

## Generiranje cesta

Budući da su ceste definirane međusobno povezanim čvorovima, na cestovnu mrežu se može gledati kao na graf u kojem su dva vrha povezana ako postoji put u kojem su ta dva vrha uzastopna. U tom grafu svaki brid se gleda kao jedan segment. Svaki segment sastoji se od šest vrhova koji čine dvije stranice. Originalni čvorovi čine dva središnja vrha, a uz to se generiraju po dva vrha s lijeve i dva vrha s desne strane segmenta kojima se određuje širina ceste. Vrhovi se poredaju u ciklus u smjeru obrnutom od kazaljke na satu i takva struktura tvori jedan segment u kojem svaki vrh zna koji od susjednih vrhova mu je prethodni, a koji sljedeći.



Slika 1. - Primjer dvaju povezanih segmenata. Narančasto označeni vrhovi su središnji vrhovi koji su određeni OSM čvorovima

Nakon što se generiraju svi segmenti u cestovnoj mreži, slijedi povezivanje segmenata kako bi se stvorio jedinstveni povezan objekt. Povezivanje se izvodi tako da se povuku pravci kroz bočne bridove segmenata i odrede presjecišta bridova na susjednim segmentima. Ovdje dolazi do izražaja činjenica da su svi segmenti definirani ciklusima vrhova u smjeru obrnutom od kazaljke na satu, jer se za svaki čvor traži presjecište prethodnog brida jednog segmenta s sljedećim bridom drugog segmenta. Ovakav način računanja presjecišta omogućuje jednostavan izračun čak i kad se u istome čvoru sijeku tri ili više segmenata. U tom slučaju, odabire se presjecište koje je najbliže prethodnom vrhu segmenta.

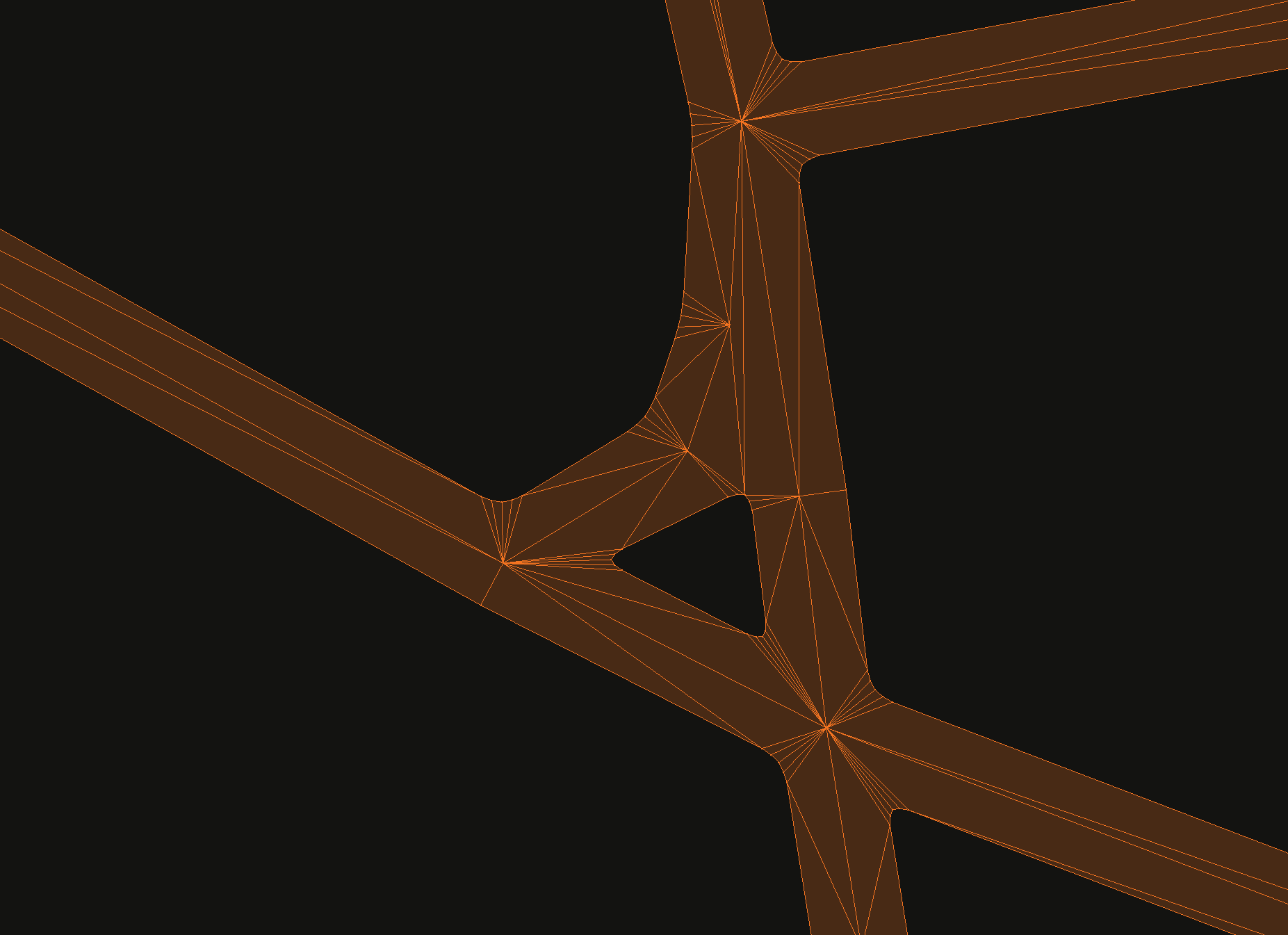
Nakon što se izračunaju presjecišta, vrhovi segmenata koji odgovaraju tim bridovima se ujedinjuju u jedan vrh točno na poziciji izračunatog presjecišta. Uz to, dodaju se novi vrhovi kako bi se dobio zaobljeni oblik ceste bez oštrih rubova. Na kraju se obavlja triangulacija svih vrhova kako bi se vrhovi povezali u stranice koje čine površinu ceste.

Na slici 2 vidi se konačan rezultat generiranja cesta.

## Generiranje rubnika

Rubnici se generiraju uz vanjski rub cesta. Prilikom generiranja svake ceste, osvježavaju se pokazivači vrhova segmenata tako da uvijek pokazuju na sljedeći vanjski vrh segmenta. Prilikom stapanja dva vrha u presjecište bridova, svaki vrh kao prethodnika (sljedbenika) dobiva prethodni (sljedeći) vrh onog vrha s kojim se stopio.

Kada su ceste izgrađene, ovakvim postupkom dobiva se niz ciklusa koji čine rubove cesta. Ti ciklusi se pretvore u krivulje po kojima se može duplicirati proizvoljan model rubnika.



Slika 2. – Kompleksno križanje generirano upotrebom opisanog algoritma

## Generiranje oznaka i imena ulica

Na svakoj cesti se uz rubnike generiraju isprekidane linije po sredini ceste i imena ulica. Za prikaz imena ulica generiraju se pravokutnici na koje se lijepi tekstura sa slikom imena ulice. Slike se automatski generiraju na temelju informacija o cestama dostupnih u OSM podacima. Slike se generiraju upotrebom programa ImageMagick koji nudi programsko sučelje za manipulaciju slikama.

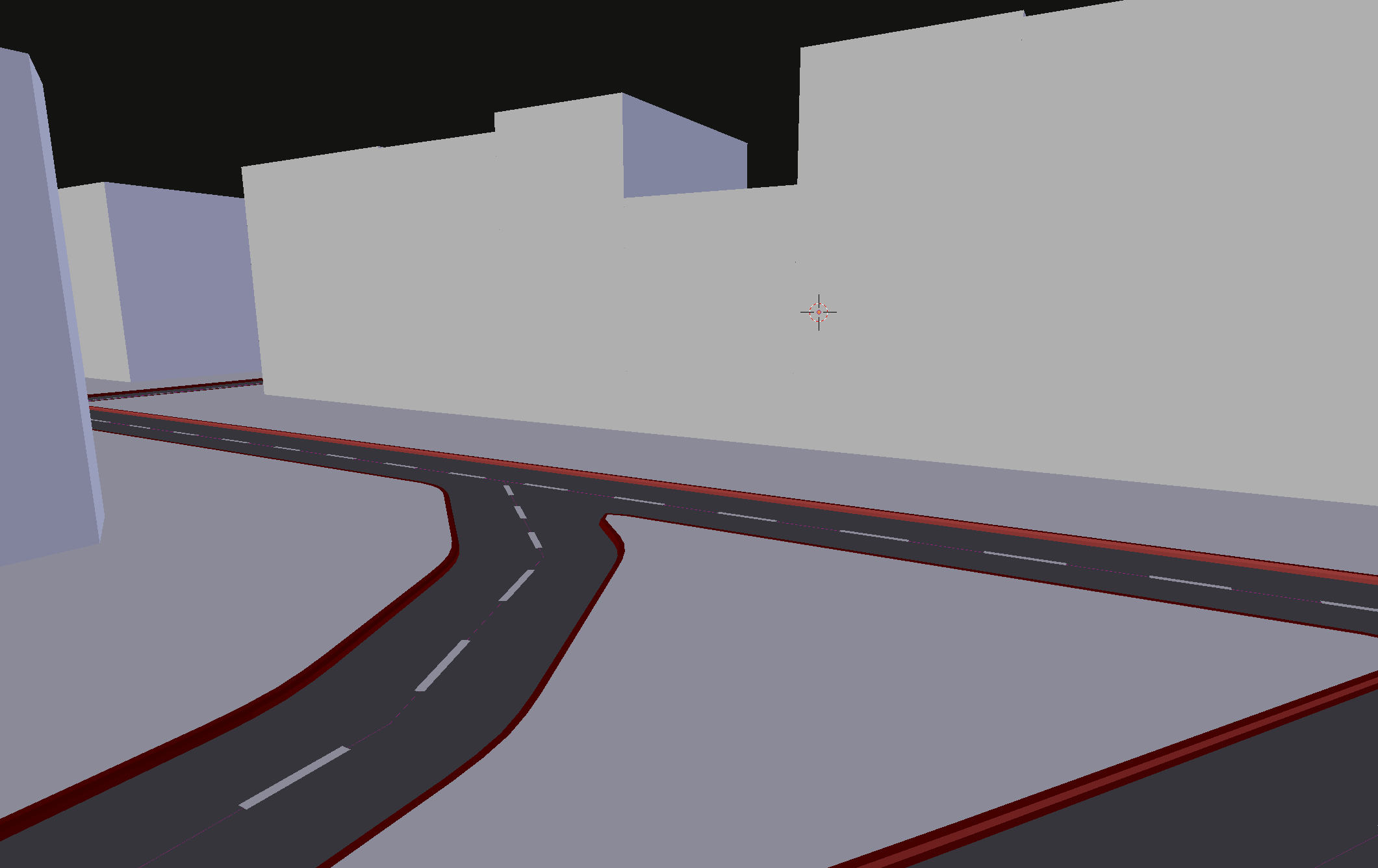
## Generiranje zgrada

Svaka zgrada definirana je ciklusom čvorova koji čine tlocrtni obris zgrade. Modeli zgrada generiraju se tako da se generira ciklus pravokutnika prema tim čvorovima, nakon čega se gornja stranica popuni triangulacijom po svim vrhovima gornjeg ciklusa. Visina svake zgrade određuje se nasumično, budući da podaci o stvarnoj visini zgrada većinom nisu dostupni.

## Problemi

Zbog načina na koji funkcionira generiranje cesta, može doći do poteškoća ako su segmenti vrlo kratki ili nedovoljno međusobno udaljeni. Tada se pojavljuju artefakti koji onemogućuju ispravan rad algoritma i može doći do greške prilikom izvođenja. Dobra stvar kod OSM datoteke je što je relativno jednostavno pronaći i prepoznati takve problematične segmente i ukloniti ih prije izvođenja.

Upravo iz tog razloga, prije generiranja segmenata se provjerava svaki brid početnog grafa. Ako je brid prekratak za zadanu širinu ceste, taj brid se uklanja i stapa s manjim od dva susjedna brida. Ovim postupkom znatno se povećava uspješnost algoritma, no i dalje nije potpuno otporan na probleme koji su uzrokovani loše definiranim podacima.



Slika 3. – Konačan rezultat generiranja grada

# Upute za korištenje

Za korištenje aplikacije potrebno je instalirati Blender (verzija 2.78) i ImageMagick (verzija 6.9.3). U Blenderovom instalacijskom direktoriju u poddirektorij */scripts/addons* potrebno je smjestiti programski kod razvijenog rješenja, nakon čega se dodatak mora omogućiti u postavkama Blendera (*preferences* > *addons*).

Prilikom instalacije ImageMagicka, potrebno je uključiti opciju instalacije razvojnog sučelja za C/C++, te instalirati Python biblioteku Wand koja omogućuje upravljanje ImageMagick sučeljem iz Pythona. Ova biblioteka mora biti dostupna iz Blendera kako bi učitavanje ispravno radilo.

Datoteke s OSM podacima zatim se jednostavno mogu učitati pritiskom na *File* > *Import > OpenStreetMap file (.osm)*. Prilikom učitavanja korisnik može podesiti postavke učitavanja koje određuju koji će sve elementi biti generirani i na koji način.

# Literatura

[1] https://www.blender.org/api/blender\_python\_api\_2\_78\_release/

[2] https://wiki.openstreetmap.org/wiki/Main\_Page

[3] https://docs.wand-py.org/en/0.4.4/

Nadopunjavanje korisničkog skeletalnog sustava u virtualnoj stvarnosti

Projektna dokumentacija

Verzija 1.0

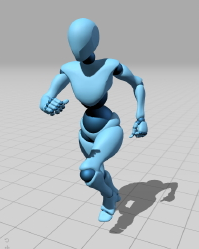
Studentski tim: Bojan Lovrović

**Nastavnik:** prof. dr. sc. Željka Mihajlović

Tehnička dokumentacija

# Opis razvijenog proizvoda

Razvijeni proizvod je programska podrška za nadopunjavanje skeletalnog sustava korisnika u virtualnoj stvarnosti, gdje nam nije poznato cijelo stanje u kojem se nalazi skelet korisnika, već samo određene kosti. Iako se radi o toliko malom broju poznatih kostiju, o istima imamo na raspolaganju preciznu poziciju i rotaciju u prostoru u bilo kojem vremenskom trenutku. Ovaj programski paket nadopunjuje cijeli, skeletalni sustav korisnika koji koristi virtualnu stvarnost, koristeći inverznu kinematiku, direktnu kinematiku, kao i apriorno poznavanje korisničkog skeleta. Za prezentaciju ovih mogućnosti korisnik se nalazi u uređenoj sceni s nekoliko estetskih dodataka kao i funkcionalnih. Primjer su kocke koje se nalaze na stolu s kojima je moguće ulaziti u interakciju. U ovoj prezentacijskoj sceni korisnik preuzima kontrolu nad modelom koji prikazuje Slika 1.



**Slika 1.** - Model korisnikovog avatara.

# Tehničke značajke

## Korištene tehnologije

* Unreal Engine 4 (grafički pogon za izradu računalnih igara)
* HTC Vive (oprema za korištenje virtualne stvarnosti)

Oba proizvoda su predviđena da rade jedno s drugim tako da je pokretanje sustava poprilično jednostavno, te su više manje sve mogućnosti dobro dokumentirane. Bitno je još napomenuti da su modeli i animacije korišteni u konačnoj verziji preuzeti besplatno sa Mixamo ([*www.mixamo.com*](http://www.mixamo.com))*.*

## Inicijalne postavke

Za potrebe uklanjanja neispravnosti, kao i provjere radi li sve kako treba, trebalo je iscrtati oba HTC Vive kontrolera, za lijevu i desnu ruku. Tu sam koristio model koji se nalazi u instalacijskim datotekama drivera za sam uređaj te je samo bilo potrebno podesiti sjenčatelje (engl. *shader*) u UE4 kako bi se isti prikazivao na ugodan način. Ovi modeli se ne vide u konačnom proizvodu i prezentaciji.



**Slika 2.** - Modeli HTC Vive kontrolera u inicijalnim postavkama

## Razred VRControlledCharacter

U razredu *VRControlledCharacter* koji nasljeđuje razred *ACharacter*, nalazi se sav C++ kod koji kontrolira poziciju, orijentaciju i animaciju (sa svim modifikacijama) modela na temelju tri matrice koje dobiva iz simulacije u koji se nalazi. Te matrice su:

* Željena transformacija glave (engl. *desired head transfrom*)
* Željena transformacija lijeve šake (engl. *desired left fist transfrom*)
* Željena transformacija desne šake (engl. *desired right fist transfrom*)

Tijekom simulacije ove vrijednosti se samo prosljeđuju izravno iz HTC Vive sučelja, te u ovom dijelu koda direktno utječu na dva svojstva: rotacija i pozicija modela u prostoru.

Pozicija se izvlači iz željene transformacije glave te se dobiva s pozivom UE4 funkcije *MoveInput()* gdje se kao parametar šalje vektor razlike između trenutne pozicije i željene.

Rotacija modela se također ažurira na temelju željene transformacije glave uz dvije razlike. Prva je da se uzima smjer gledanja glave, a ne njena pozicija, te se na temelju tog vektora rotira model. Druga je da se uzimaju vektori smjera relativnih pozicija šaka u odnosu na poziciju glave, te se gleda koliko su ti vektori slični vektoru smjera glave. Ako su vrlo slični, to znači da su se ruke i glava zarotirali u prostoru dakle vjerojatno se je i korisnikovo tijelo, pa se rotira cijeli model maksimalnom brzinom. Ako je korisnik samo pogledao u stranu (nije rotirao ostatak tijela) to će se vidjeti po vektorima šaka te će se cijeli model rotirati u smjeru glave minimalnom brzinom. Ovo naravno ne uključuje samu kost glave u skeletalnom modelu koja se rotira jednakom brzinom kao što se ažurira željena transformacija glave.

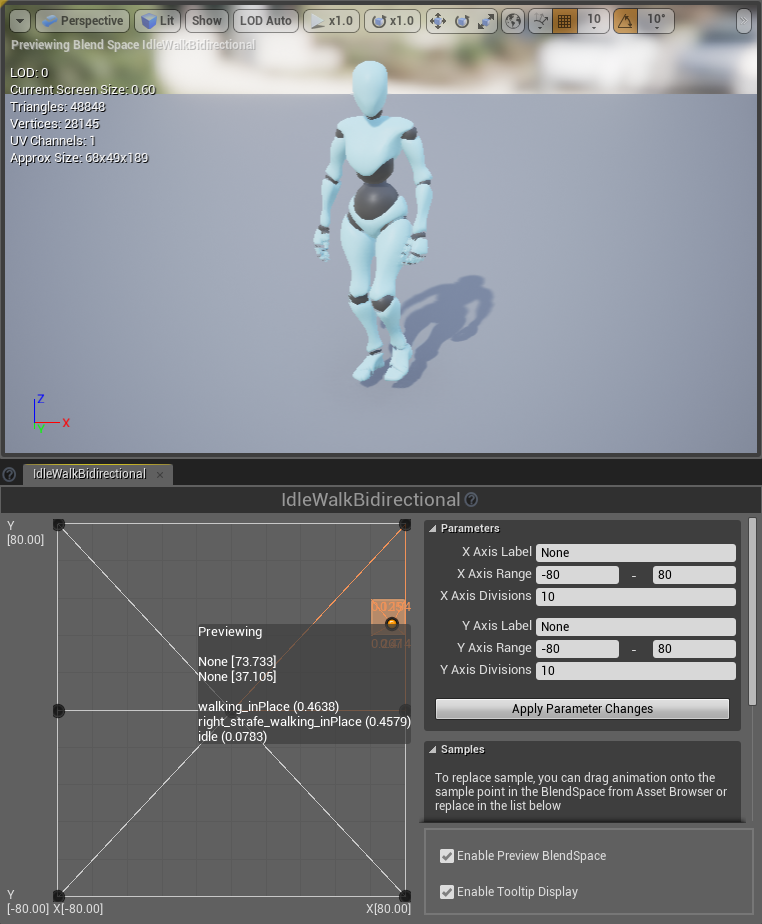
## Miješanje animacija

Sustav je izgrađen da se od točke A do točke B može doći rotacijom prema odredištu te gibanjem naprijed, kao i gibanjem pod bilo kojim kutem bez rotiranja. Budući da je prvi način samo pojednostavljenje drugog načina (ako razmatramo samo animacije gibanja) promotrit ćemo samo drugi.

Dakle potrebno je bilo omogućiti animacije za gibanje pod bilo kojim kutem u odnosu na trenutačni vektor smjera modela. To je postignuto uz pomoć UE4 podrške za miješanje animacija. Da bi se taj sustav pokrenuo na željeni način učitane su četiri animacije za model:

* Hodanje naprijed
* Hodanje nazad
* Hodanje lijevo
* Hodanje desno

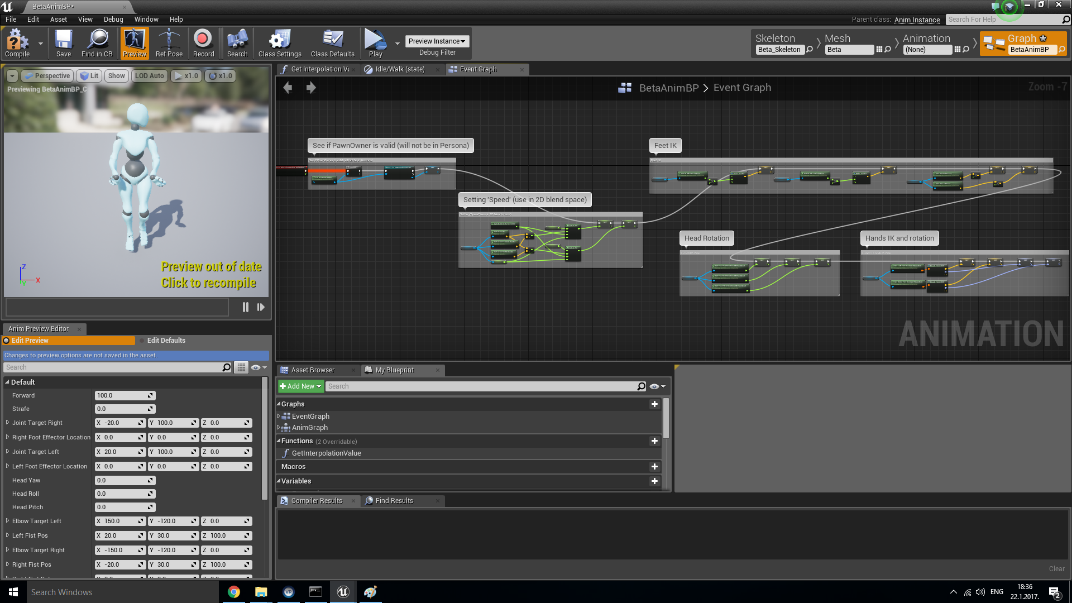
Miješanje animacija se odvija kao što prikazuje Slika 3. Koristi se baricentrična interpolacija kako bi se dobile jačine pojedine animacije za dani vektor smjera.



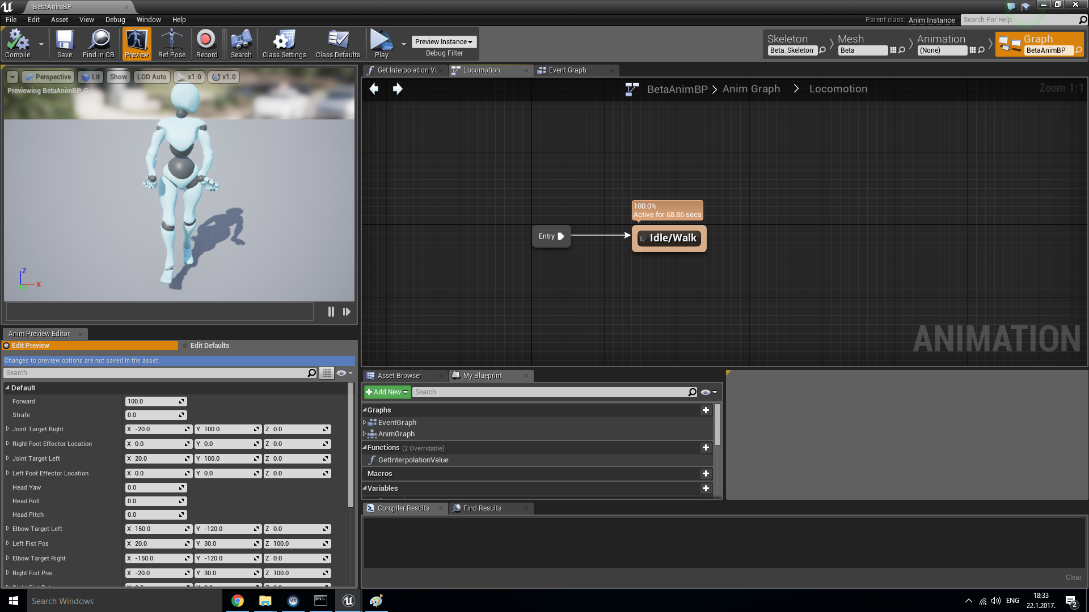
**Slika 3.** - Miješanje animacija

## Inverzna kinematika

Upotreba inverzne kinematike je prisutna u nogama, kao i rukama, te je u potpunosti izrađena u UE4 shematskim planovima (engl. *blueprint*). Konkretan tip shematskog plana koji se je koristio za ovu primjenu se naziva Animacijski shematski plan (engl. *Animation Blueprint*). Sastoji se od dva dijela: klasičnog UE4 shematskog plana te automata stanja (engl. *state machine*), kao što se vidi na slikama 4 i 5.



**Slika 4.** - Klasičan shematski plan u animacijskom shematskom planu



**Slika 5.** - Automat stanja u animacijskom shematskom planu

Dvije je varijable potrebno objasniti prije ulaska u rješenje. To je lokator učinka (engl. *effector location*) ili ciljna pozicija kosti koju animiramo inverznom kinematikom, te lokacija ciljanog zgloba (engl. *joint target location*) ili smjer u kojem će zglob kosti (spojnica roditelja i pra-roditelja kosti) težiti. Prioritet je naravno na ciljnoj poziciji te je radi toga zglob kosti rijetko u lokaciji ciljanog zgloba.

U nogama služi samo da prilikom animacije pomakne stopala prema gore kada model "čučne" u lokalnom prostoru. Ovdje je lokacija ciljanog zgloba postavljena tako da uvijek bude ispred nožnih prstiju tako da se koljeno uvijek savija na pravu stranu, dok je lokator učinka ostavljen kakav bi trebao biti iz animacije uz (po potrebi) mijenjanu y komponentu.

U rukama je lokacija ciljanog zgloba lakat te je uvijek postavljen na lijevu i desnu stranu (ovisno u ruci) tako da se laktovi šire od tijela. Svaka šaka je lokator učinka te uzima poziciju od HTC Vive kontrolera. Primjer ovoga je dan na Slici 2 gdje korisnik gleda svoje šake.

## Unaprijedna kinematika

Jednom kada je cijeli model na odgovarajućem mjestu, ispravno rotiran, te kada su šake postavljene u pravi položaj, potrebno je primijeniti zadnju modifikaciju skeleta modela. Potrebno je rotirati glavu i šake tako da orijentacija odgovara pripadajućim uređajima.

# Upute za korištenje

1. Instalirati Unreal Engine 4
2. Po mogućnosti ažurirati pokretačke programe (engl. *driver*)
3. Spojiti HTC Vive uređaj s računalom.
4. Pokrenuti "*Room set up*" kako bi se inicijalizirao prostor po kojem će se korisnik kretati.
5. Staviti HDM (engl. *head mounted display*) na glavu te uzeti oba kontrolera.
6. Pokrenuti programski paket preko Unreal Engine 4 uređivača.

# Literatura

[1] D. H. Eberly, (2010) Game Physics, 2nd ed., Elsevier

[2] I. Millington, (2007), Game Physics Engine Development, Elsevier

[3] C. Ericson (2004), Real-Time Collision Detection, Elsevier

**Odgođeno fizikalno temeljeno sjenčanje**

**Projektna dokumentacija**

**Verzija 1.0**

Studentski tim: Bruno Pregun

Stephanie Cheng

Nastavnik: prof. dr. sc. Željka Mihajlović

# Tehnička dokumentacija

# Opis razvijenog proizvoda

Izrađeni program demonstrira kombinaciju dviju tehnika koje se koriste za efikasan i fotorealističan prikaz objekata u virtualnoj sceni. Program je napisan u C++ programskom jeziku i koristi OpenGL za prikaz 3D grafike. Prva korištena tehnika je odgođeno sjenčanje koje dozvoljava smanjenje kompleksnosti virtualne scene, a druga tehnika je fizikalno temeljeno sjenčanje koje omogućuje realističniji prikaz materijala i osvjetljenja.

# Tehničke značajke

## Odgođeno sjenčanje

U uobičajenim (unaprijednim) načinima sjenčanja objekti se sjenčaju tako da se za svaki pojedini objekt gleda kako svako pojedino svijetlo utječe na njegovo osvjetljenje. To je vrlo ne efikasno jer količina računanja ovisi o umnošku broja objekata i broja svjetala u sceni. Pomoću odgođenog sjenčanja pokušavamo zaobići tu neefikasnost. Odgođeno sjenčanje dijeli proces sjenčanja u dvije faze i između njih koristi međuspremnik (tzv. G-buffer).

Prva faza gradi međuspremnik tako da ga puni svim informacijama koje su potrebne da bi se izračunalo osvjetljenje scene. Program puni međuspremnik za svaki objekt u sceni. Za to se koristi GLSL shader program koji umjesto da crta sliku na zaslon, pohranjuje izračunate vrijednosti u zasebni okvir slike (engl. *framebuffer*). U izrađenom programu međuspremnik se sastoji od 4 slike. U prvu sliku se pohranjuje pozicija svakog iscrtanog slikovnog elementa u 3D prostoru, odnosno u RGB kanale slike spremaju se x, y i z koordinate. U drugu sliku se spremaju koordinate normala svakog slikovnog elementa. U treću sliku zapisuje se informacija o osnovnoj (neosvijetljenoj) boji materijala koji se iscrtava. U četvrtu i posljednju sliku pohranjuje se informacija o vrsti materijala koji se iscrtava. U crveni kanal posljednje slike zapisuje se je li materijal metal ili nemetal, a u zeleni hrapavost materijala. Ti parametri će se kasnije koristiti za fizikalno temeljeno sjenčanje.

Druga faza koristi izrađeni međuspremnik za sjenčanje i iscrtavanje same scene. Za svako svijetlo iscrtava se jedan pravokutnik koji pokriva zaslon. Kao ulazne vrijednosti shader programa koristi prethodno izrađene slike u međuspremniku. Svaki izvor dati će svoj doprinos svjetla scene i ti se doprinosi akumuliraju, čime se dobiva finalni prikaz.

## Fizikalno temeljeno sjenčanje

Fizikalno temeljeno sjenčanje (eng. *Phisically based rendering* – PBR), je način sjenčanja koji približno oponaša osvjetljenje stvarnog svijeta. To se postiže korištenjem fizikalnih formula ponašanja svjetla i time oponaša svijetlo na realističniji način od starih tehnika sjenčanja poput Phong ili Blinn-Phong.

Također jedna od prednosti jest, pošto oponaša stvarnu fiziku svjetla, razni materijali će izgledati korektno u danom virtualnom okruženju bez potrebe dodatnog podešavanja postavki materijala.

Fizikalno temeljeno sjenčanje bazira se na fizikalnim principima modela mikro-površina, očuvanje energije i jednadžbi sjenčanja.

Model mikro-površina govori kako se svaka površina može opisati kao nakupina malih ogledala, i ovisno o grubosti površine ta ogledala su drugačije poravnata. Što je površina grublja to su ta ogledala više kaotično usmjerena. Ovaj koncept je važan za refleksiju svjetla na raznim vrstama površinama, grube površine zbog kaotične organizacije ogledala svjetlo će se odbijati u puno različitih kuteva i time ju raspršiti, dok kod glatkih površina većina zraka svjetlosti će se odbiti po zakonu refleksije svjetla i time će se materijal ponašati više nalik zrcalu.

Drugi važni temelj fizikalno temeljenog sjenčanja je zakon očuvanja energije, koji kaže da energija odbijenog svjetla ne može biti veća od energije dolaznog svjetla. Upadna zraka koja udari u površinu raspadne se na reflektiranu i refraktiranu zraku. Reflektirana zraka se odbije, a refraktirana zraka prolazi kroz površinu. Ako refraktirana zraka preda svoju energiju kolizijama s česticama materijala objekta, tu zraku smatramo „apsorbiranom“ i time se gubi njena svjetlosna energija. Refraktirana zraka se također može odbiti unutar predmeta i raspršiti se pod raznim kutovima (engl. *subsurface scattering*), ali u ovoj implementaciji fizikalno temeljenog sjenčanja mi pretpostavljamo da su sve refraktirane zrake apsorbirane.

Metali i nemetali se tretiraju drugačije kod fizikalno temeljenog sjenčanja zbog toga što metali upijaju sve refraktirane zrake i time ne pokazuju difuzne boje.

Trenutno najbolji model za simuliranje ponašanja svjetlosti je formula koja se zove jednadžba sjenčanja. Fizikalno temeljeno sjenčanje se bazira na aproksimaciji te formule. Tu aproksimaciju nazivamo jednadžba refleksije.

Jednadžba refleksije sumira zračenja svih zraka koje ulaze u smjeru **ωi** po cijeloj polu kugli **Ω** pomnoženih s faktorom **fr** koje ulaze u točki **p** i vraća tu sumu reflektiranog svjetla **Lo** u smjeru promatrača.

Simbol **fr** predstavlja funkciju dvosmjerne reflektivne distribucije odnosno BRDF (eng. *bidirectional reflective distribution function*). Ta funkcija skalira dolazno zračenje u odnosu na osobine materijala. Za potrebe ovog projekta korištena je Cook-Torrance varijanta te funkcije.

BRDF formula je, kao i upadne zrake, podijeljena na dio koji se refraktira i dio koji se reflektira. Parametar **ks** određuje omjer zraka koje se reflektiraju, a njemu inverzan parametar **kd** omjer zraka koje se refraktiraju, odnosno apsorbiraju. Radi prethodno spomenutog zakona očuvanja energije zbroj ta dva parametara mora biti jednak 1.

Faktor flambert je difuzna komponenta i određen je bojom površine objekta, a faktor fcook-torrance je spekularna komponenta i određen je sljedećom formulom:

Simboli D, F i G predstavljaju funkcije koje aproksimiraju specifični dio reflektivnih svojstava površine.

D je funkcija normalne distribucije koja daje približnu količinu mikro-površina koje su usmjerene prema polovičnom vektoru (vektoru na polovici između vektora svijetla i vektora pogleda). Ova funkcija ovisi o hrapavosti materijala i glavna je komponenta za aproksimiranje mikro-površina.

G je funkcija geometrije i ona opisuje svojstvo da jedne mikro-površine prekriju druge, time stvaraju sjenu i smanjuju količinu reflektiranog svjetla. F je Fresnel funkcija koja opisuje omjer refleksije u ovisnosti o kutu površine.

Svaka od ovih funkcija je aproksimacija fizikalnog svojstva i postoje mnogo raznih varijanti svake od njih. Nekima je cilj realističnost prikaza, a nekima efikasnost izvođenja. U ovom projektu korištene su sljedeće varijante:

Trowbridge-Reitz GGX ( α je parametar hrapavosti ):

Smith's Schlick-GGX ( k = 0.125⋅(α+1)2 ):

Fresnel-Schlick ( F0 je 0.04 za nemetale, dok kod metala ovisi o njihovoj boji ):

# Upute za korištenje

Program se pokreće izvršnom datotekom. Nakon pokretanja otvara se prozor za prikaz scene. Držanjem pritisnute lijeve tipke miša i pomicanjem miša unutar prozora omogućeno je okretanje kamere oko središta scene. Na sličan način, samo pomoću desne tipke miša može se okretati i objekt u sceni.

Program učitava model iz *.obj* datoteke. Za korištenje fizikalnog sjenčanja programu su potrebne dvije teksture u *.dds* formatu. Prva tekstura definira difuznu boju predmeta (albedo), a druga parametre materijala (crveni kanal – metal/nemetal, zeleni kanal - hrapavost).

# Literatura

[1] <https://en.wikipedia.org/wiki/Deferred_shading>

[2] <https://www.marmoset.co/toolbag/learn/pbr-theory>

[3] <http://blog.wolfire.com/2015/10/Physically-based-rendering>

[4] <https://www.allegorithmic.com/pbr-guide>

[5] [http://blog.selfshadow.com/publications/s2016-shading-course](http://blog.selfshadow.com/publications/s2016-shading-course/)

[6] <https://learnopengl.com/#!PBR/Theory>

[7] <http://www.codinglabs.net/article_physically_based_rendering.aspx>

[8] <https://en.wikipedia.org/wiki/Rendering_equation>

[9] <http://freepbr.com/>

Generiranje mekih sjena u prostoru prikaza

Projektna dokumentacija

**Verzija 1.0**

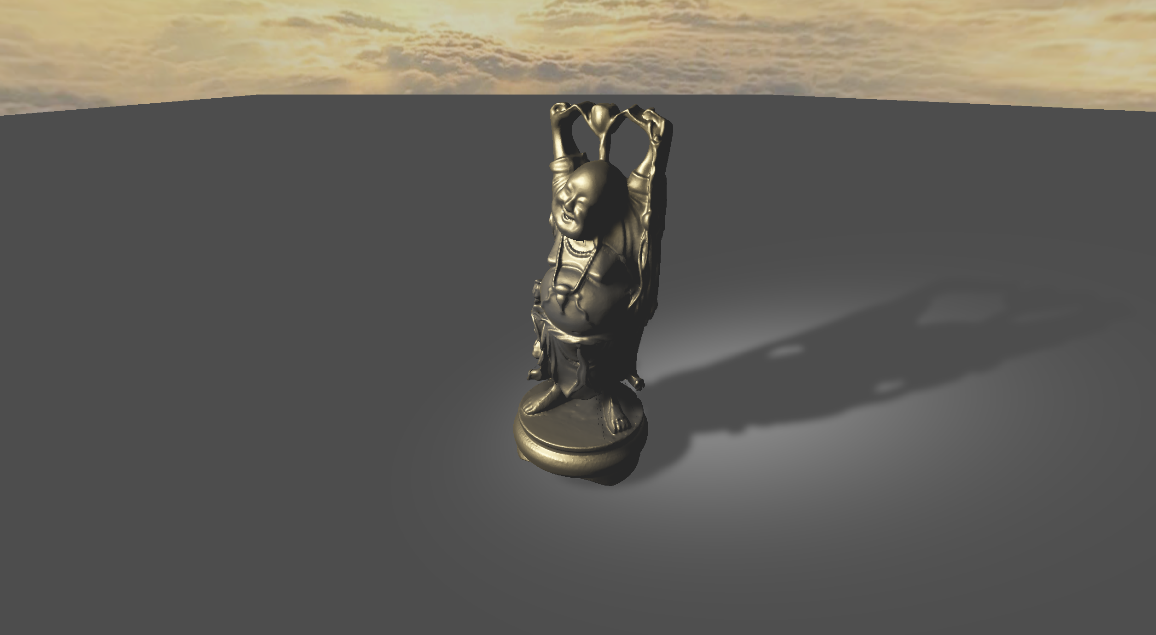
Studentski tim: Jure Ratković

Nastavnik: prof. dr. sc. Željka Mihajlović

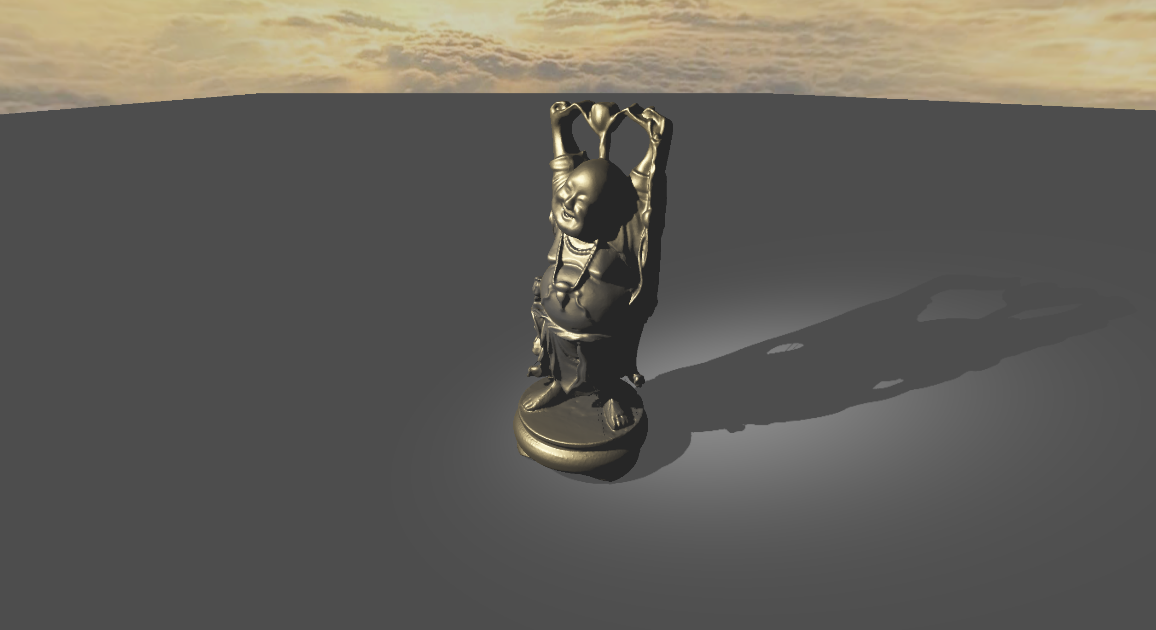
Tehnička dokumentacija

# 1. Opis razvijenog proizvoda

Razvijen je pokretač s odgođenim iscrtavanjem (eng. *deferred rendering engine*) u kojem je implementirana problematika projekta. Razvijena je aplikacija koja koristi spomenuti pokretač, te koja nudi interaktivne značajke putem tipkovnice i miša. Aplikacija učitava model te gradi jednostavnu scenu koja se sastoji od učitanog modela, ravnine na kojoj model stoji te izvora svijetla (Slika 1. i 2.). Kamera i izvor svijetla se mogu pomicati, te se generiranje mekih sjena u prostoru prikaza može uključivati i isključivati po potrebi.



**Slika 1**. - Scena s mekim sjenama



**Slika 2.** - Scena bez mekih sjena

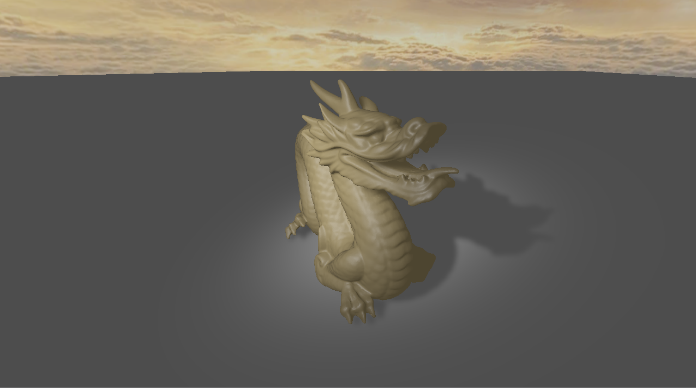
# 2. Tehničke značajke

Postupak generiranja mekih sjena u prostoru prikaza:

1. Generiranje mape sjena – iscrtavaju se objekti iz perspektive svakoga svjetla te se sprema dubina svakoga piksela u mapu sjena svakoga svjetla
2. Projiciranje sjena – sjene svakoga svjetla se projiciraju u prostor pogleda. Koristi se zaseban međuspremnik za svaku mapu, te se pri projiciranju može koristiti PCF za bolje rezultate
3. Zamućivanje projiciranih sjena – međuspremnici projiciranih sjena se zamućuju Gaussovom jezgrom (eng. *Gaussian blur*). Širina jezgre ovisi o udaljenosti piksela od kamere te odaljenosti piksela od bacača sjene
4. Primjena sjena na scenu – rezultat sjenčanja piksela se pomnoži s vrijednosti istoga u zamućenoj projiciranoj mapi sjena



**Slika 3.** - Mapa sjena S**lika 4.** - Projicirana mapa sjena



**Slika 5.** - Zamućena projicirana mapa sjena **Slika 6.** - Primijenjena mapa sjena

# 3. Upute za korištenje

Aplikacija se pokreće putem dane izvršne datoteke. Odgovarajuće scene se pokreću pomoću zastavica komandne linije "*-shadows -dragon*" te "*-shadows -buddha*". Kada se scena učita, kamera i svijetlo se kontroliraju pomoću miša, a uključivanje i isključivanje mekih sjena se vrši pritiskom tipke „b“.

# 4. Literatura

[1] http://http.developer.nvidia.com/GPUGems3/gpugems3\_ch08.html

[2] http://http.developer.nvidia.com/GPUGems/gpugems\_ch11.html

[3] https://imgtec.com/blog/implementing-fast-ray-traced-soft-shadows-in-a-game-engine/

[4] http://www.gamedev.net/page/resources/\_/technical/graphics-programming-and-theory/soft-edged-shadows-r2193

Ispitivanje mogućnosti pokretačkog sustava Unity3D za razvoj korisničkih alata za generiranje multimedijskih sadržaja

Projektna dokumentacija

**Verzija 1.0**

Studentski tim: Zrinka Lekić

Nastavnik: doc. dr. sc. Siniša Popović

Tehnička dokumentacija

# 1. Opis razvijenog proizvoda

Napravljena je analiza mogućnosti prikaza videomaterijala unutar pokretačkog sustava Unity3D. Ispitane su razne opcije za prikaz videozapisa, te je izrađena aplikacija za ispitivanje prikaza videozapisa kojom su se provela mjerenja. Na temelju mjerenja utvrđena je najprikladnija besplatna tehnologija za prikaz videomaterijala unutar korisničkih alata za generiranje multimedijskih podražaja zasnovanim na pokretačkom sustavu Unity3D.

Osim aplikacije za ispitivanje prikaza videomaterijala, razvijena je aplikacija za interaktivni prikaz podražajne sekvence.

Aplikacija za ispitivanje prikaza videomaterijala mjeri brzinu prikaza sličica po sekundi (engl. *framerate*) za svaki okvir koji se prikazuje na ekranu. Mjerenja se spremaju u tekstualnu datoteku kako bi se ti podaci u vanjskom programu iskoristili za vizualizaciju i kvantitativnu usporedbu različitih načina prikaza videomaterijala.

Unutar aplikacije za interaktivni prikaz podražajne sekvence implementirano je korisničko sučelje i potrebna arhitektura za prikaz podražaja (video, slika, zvuk). Odabirom podražaja na sučelju ono se trenutno prikazuje na drugom ekranu namijenjenom ispitaniku.

# 2. Tehničke značajke

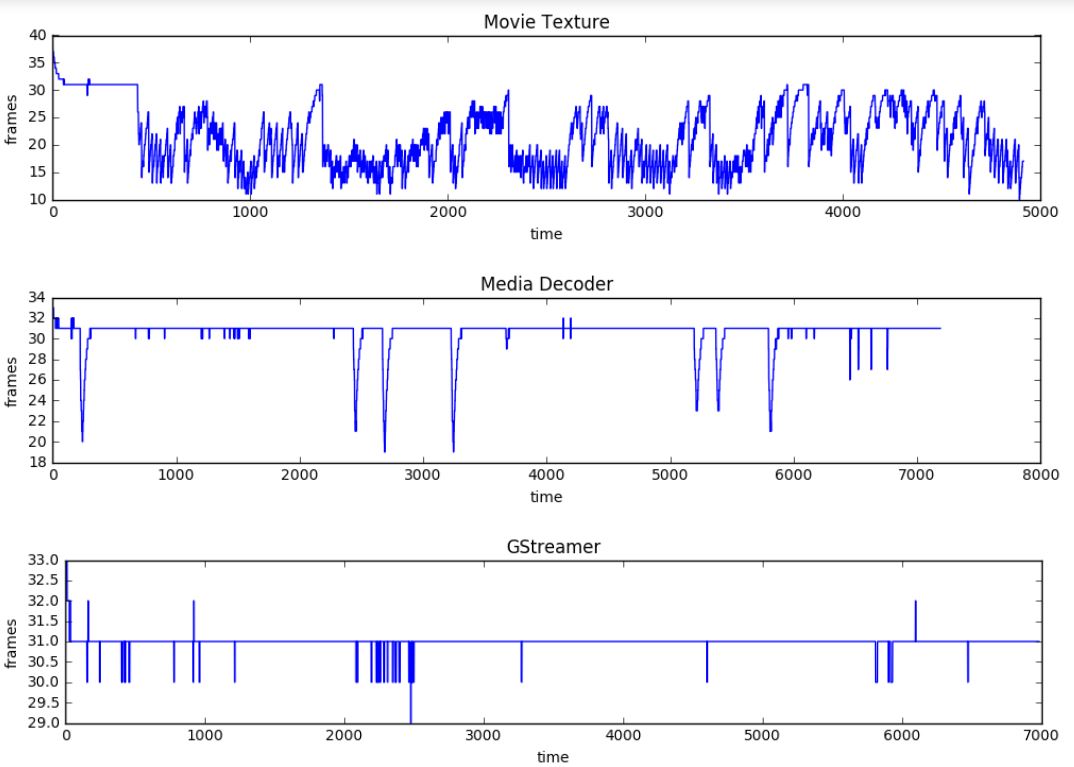
Aplikacija je prvenstveno namijenjena izvođenju na operacijskom sustavu Windows, no može se pokrenuti i na sustavima Linux i MacOS. Nadalje, aplikacija implementira više modula za prikaz videomaterijala koji su bili namijenjeni ispitivanju. Moduli koje se može koristiti su: Movie Texture, Media Decoder, GST Movie Texture, VLC Player i WWW modul. Aplikacija kroz ove module podržava sve raširene formate za prikaz videomaterijala poput formata mp4, mkv, ogv i drugih, ovisno o modulu.

Moduli dohvaćeni iz Unitijevog dućana (*Asset store*) Media Decoder i GST Movie Texture koriste programsko okružje (engl. *framework*) otvorenog koda GStreamer za dekodiranje i prikaz videozapisa.

Za potrebe ispitivanja i mjerenja kvalitete reprodukcije videozapisa korišten je videozapis „Big buck bunny“ preuzet iz [2] napravljen softverom otvorenog koda Blender. Navedeni videozapis dostupan je za preuzimanje u različitim rezolucijama i načinima kodiranja. Za mjerenja korišten je videozapis rezolucije 1080p kodiran formatom h264.

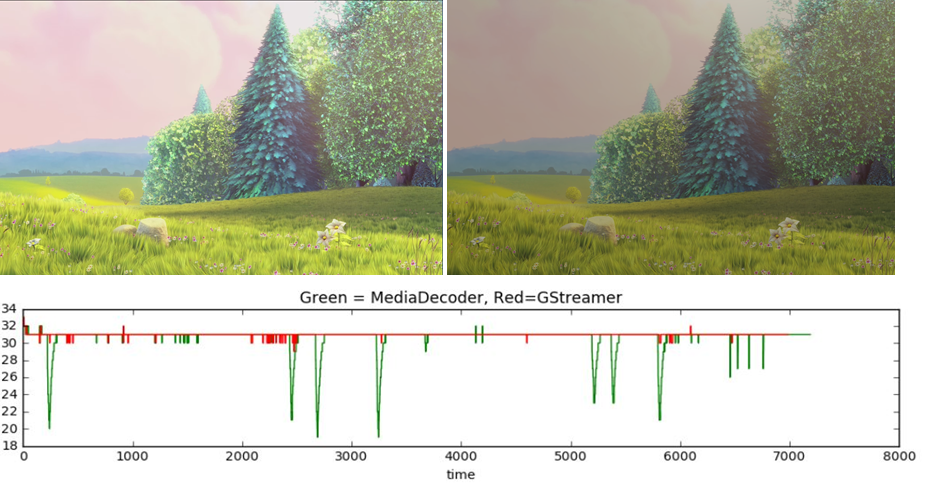
S obzirom na željene karakteristike modula za prikaz videozapisa, nisu testirani moduli za prikaz VLC-om niti WWW modul. Naime, pozivanjem vanjskog procesa za prikaz napušta se prozor Unitijeve aplikacije te se time gubi mogućnost točnog mjerenja, a i finija kontrola nad prikazom videozapisa (npr. prekidanje na točno desetoj sekundi videozapisa). WWW modul trenutno podržava samo reprodukciju zapisa s interneta, što nije u skladu sa željenim značajkama krajnjeg korisničkog alata.

Slika 1. prikazuje rezultate mjerenja. Iz grafova je jasno vidljivo da modul MovieTexture koji implementira Unitijevu klasu MovieTexture za prikaz videomaterijala nije pogodan za prikaz videozapisa tako visoke rezolucije zbog očitih padova u broju prikazanih sličica u sekundi na ekranu.

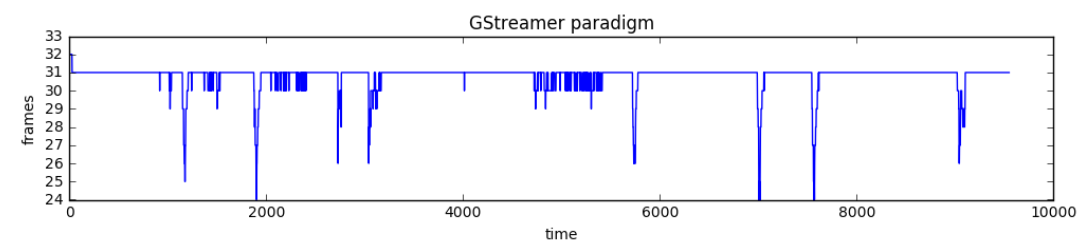


Slika 1. - Rezultati mjerenja modula za prikaz videozapisa

Nadalje, vidljivo je da je modul GStreamer, implementiran pomoću modula GST Movie Texture, ima bolje performanse od modula MediaDecoder – ima manje padove u broju sličica po sekundi na pojedinim dijelovima paradigme. Iz ovog razloga, za implementaciju unutar aplikacije za interaktivni prikaz podražaja implementiran je i za prikaz se koristi modul GStreamer. Slika 2. prikazuje drugu razliku između ova dva modula – razlika u svjetlini slike, iako su testirani pod istim scenskim uvjetima. Ispod prikaza razlike boja, superponirani su grafovi mjerenja prikaza ovih dvaju modula, te je vidljivo da modul MediaDecoder ima nešto lošije performanse u usporedbi s modulom GStreamer.

**Slika 2.** - Usporedba prikaza modula MediaDecoder i GStreamer

S obzirom na rezultate mjerenja različitih modula, za ispitivanje performansi reprodukcije podražajne paradigme odabran je modul GStreamer. Paradigma se sastojala od videozapisa te slikovno-zvučnih podražaja. Kod Slike 3. vidljivi su trenuci učitavanja videozapisa s diska kada broj okvira pada ispod prosjeka. Budući da se oni događaju u trenutku dok se prikazuje slikovno-zvučni zapis, ta potonuća u broju prikazanih okvira nisu vidljiva oku.



Slika 3. - Rezultati mjerenja modula GStreamer pri prikazu paradigme

Unutar aplikacije za interaktivno definiranje paradigme također je za prikaz korišten modul GStreamer. Ta aplikacija pretpostavlja postojanje dvaju aktivnih ekrana koja su priključena u računalo. Korisničko je sučelje implementirano korištenjem klasa iz Unity.UI imeničkog prostora. Na Slici 4. s desne strane je sučelje koje se prikazuje eksperimentatoru koji interaktivno upravlja s podražajima. U lijevom stupcu su dostupne slike sa zvukovima, a u desnom videozapisi. S lijeve strane je ekran koji je namijenjen ispitaniku. Izrađena aplikacija omogućava eksperimentatoru da u stvarnom vremenu bira i prikazuje multimedijske podražaje ispitaniku, a u budućnosti se može proširiti i s generiranjem interaktivnih 3D stimulacija za ispitanika kao što su kognitivne igre odnosno testovi.



Slika 4. - Ekran koji se prikazuje ispitaniku i interaktivno sučelje eksperimentatora

# 3. Upute za korištenje

Za odabir modula koji će se koristiti za prikazivanje te nad kojim će se provesti mjerenja, unutar aplikacije za mjerenje brzine izvođenja videomaterijala prvo je potrebno odabrati modul za ispitivanje.

Po odabiru modula, pokreće se reprodukcija videozapisa i započinje mjerenje. Po završetku videozapisa mjerenje se sprema u tekstualnu datoteku. Nakon toga može se odabrati sljedeći modul za ispitivanje. Aplikacija se gasi pritiskom na tipku Escape.

Aplikacija za interaktivni prikaz paradigme iscrtava se na dva ekrana. Jedan se prikazuje ispitaniku, dok na drugome osoba koja upravlja eksperimentom bira koji će se podražaj prikazati ispitaniku. Podražaj se bira pritiskom na njegovu sličicu (engl. *thumbnail*). Podražaj se zatim prikazuje dok ne završi ili dok ispitivač ne pokrene sljedeći. Aplikacija se gasi pritiskom na tipku Escape.

# 4. Literatura

[1] Unity3D manual, <https://docs.unity3d.com/Manual>, pristupljeno 12. siječnja 2017.

[2] Big Buck Bunny, <https://peach.blender.org/> , pristupljeno 12. siječnja 2017.