SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1090

Upravljanje gestama 3D objektom pomoću uređaja Leap Motion

Damir Ciganović-Janković

Zagreb, lipanj 2015.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 6. ožujka 2015.

DIPLOMSKI ZADATAK br. 1090

Pristupnik: **Damir Ciganović-Janković (0036461068)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Upravljanje gestama 3D objektom pomoću uređaja Leap Motion**

Opis zadatka:

Proučiti mehanizme upravljanja gestama u virtualnom okruženju te uređaj Leap Motion i njegove mogućnosti u ovom kontekstu. Proučiti razvojni okvir Three.js koji omogućava lakšu uporabu tehnologije WebGL-a. Razraditi skup gesti koje će omogućiti manipuliranje trodimenzijskim objektom. Napraviti programsku implementaciju koja će u razvojnom okviru Three.js demonstrirati mogućnosti upravljanja gestama trodimenzijskim objektom. Načiniti ocjenu ostvarenih rezultata.
Izraditi odgovarajući programski proizvod. Rezultate rada načiniti dostupne putem Interneta. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Zadatak uručen pristupniku: 13. ožujka 2015.
Rok za predaju rada: 30. lipnja 2015.

Mentor:

Prof. dr. sc. Željka Mihajlović

Predsjednik odbora za diplomski rad profila:

Prof. dr. sc. Siniša Srbljić

Djelovođa:

Doc. dr. sc. Tomislav Hrkać

Zahvala

Zahvaljujem prof. dr. sc. Željki Mihajlović što me je pratila kao mentor tijekom preddiplomskog i diplomskog fakulteta, svojoj majci što mi je omogućila studiranje te svojoj zaručnici što mi je bila podrška za vrijeme pisanja diplomskog rada.

Sadržaj

1. Uvod

Od nastanka prvog digitalnog računala Eniac (1946.) [1], ljudi su shvatili kolike mogućnosti računalo nudi, te su uvijek tražili način kako te mogućnosti iskoristiti što bolje. Jedan od načina povećanja učinkovitosti računala je i poboljšanje interakcije čovjeka koji koristi računalo sa samim računalom. Pošto su pisaći strojevi imali već dugu povijest korištenja, prvi korak, nastanak tipkovnice, bio je lak prijelaz. Sljedeći korak bio je uporaba računalnog miša koji je omogućio razvoj grafičkog sučelja računala čime je razvoj tehnologije ubrzano počeo rasti. Od mehaničkog miša „s kuglicom“ do optičkog miša, od žičanog do bežičnog miša, bolja interakcija čovjeka i računala nije bila moguća bez razvoja novih uređaja koji će joj doprinijeti. Ideja koja, još od raznih znanstveno fantastičnih televizijskih serija, zaokuplja ljudsku maštu su uređaji čiji se unos podataka temelji na prepoznavanju ljudskih pokreta. Jedan od pokušaja izrade uređaja takve vrste je uređaj Power Glove (Slika 1.) tvrtke Nintendo [2] korišten za interakciju s igraćom konzolom tako da je slao različite signale s obzirom na položaj ruke. Unatoč ulaganju velikog novca u reklamiranje, projekt je doživio ekonomsku propast što je dovelo do zastoja u razvoju novih sličnih uređaja, no s razvojem tehnologije, posljednjih nekoliko godina dolazi novi val inovativnih uređaja koji su naišli na veliki uspjeh tamo gdje Power Glove nije.

Slika 1. Power Glove [2]

Neki od tih uređaja su primjerice Intel Real Sense [3], Kinect [4] i Leap Motion [5]. Intel Real Sense je uređaj koji omogućuje prepoznavanje položaje ruke, razne geste, izraze lica te prepoznaje čak i govor. Njegovo korištenje je zamišljeno tako da korisnik sjedi ispred računala s uređajem postavljenim na vrh monitora i okrenutim prema korisniku. Uređaj je malen i praktičan, no zbog položaja senzora

se mora pripaziti kako ruke ne bi zaklanjale pogled prema ekranu monitora. Uređaj Kinect je nasuprot Intel Real Sense-u nešto veći i skuplji. Isto tako ima mogućnost praćenja ruku i lica, ali i kostura cijelog tijela. Na početku je bio zamišljen kao pomoćni uređaj za XBOX [6] igrače konzole, no kasnije mu je prepoznata uloga i izvan industrije konzolskih igrica te se često koristi u računalnim aplikacijama različite namijene. U ovom radu obrađivat će se zadnji od spomenutih uređaja, Leap Motion. Razmotrit će se mogućnosti uređaja u prepoznavanju gesti. Osim toga, prepoznavanje gesti bit će ugrađeno i u Web aplikaciju Micro Soot maintenance app koja sadržava simulaciju održavanja uređaja Micro Soot tvrtke AVL na način da će se 3D modelom uređaja Micro Soot upravljati gestama pomoću Leap Motion kontrolera.

2. Uređaj Leap Motion

2.1. Što je Leap Motion?

Leap Motion je uređaj koji služi za praćenje pokreta ruku. To uključuje, položaj podlaktice, šake, prstiju, te predmeta koji liče na prste, a tu spadaju svi predmeti koji imaju tanje, cilindrično tijelo poput olovke ili štapića. Uređaj je napravljen od strane Indonezijske tvrtke Leap Motion, Inc koju su osnovali Michael Buckwald i David Holz [7]. Postoje dvije vrste uređaja Leap Motion. Prva vrsta služi za način korištenja kada se uređaj Leap Motion nalazi na stolu, u statičnom stanju. Druga vrsta se koristi u aplikacijama čije je područje interesa virtualna stvarnost. U tom slučaju je Leap Motion priključen na uređaj Oculus Rift [8]. Oculus Rift je uređaj koji se stavlja na glavu tako da prekriva oči te oba oka dobivaju svoj posebni prikaz čime korisnik može dublje proživjeti virtualnu scenu računalne igre ili bilo koje druge aplikacije koju koristi. Korisnik mijenja pogled u aplikaciji okretanjem glave te zbog toga Leap Motion, koji je pričvršćen na prednji dio Oculus Rift-a, neće biti u statičnom položaju, već će se kretati svaki put kada korisnik pomakne glavu. U ovom radu proučavat će se prva vrsta uređaja.

2.1.1. Specifikacije uređaja

Uređaj je malen s obzirom na ostale uređaje slične namijene. Dugačak je 80 mm, širok 30 mm, visok 11.25 mm te vrlo lagan (32.4 grama) što ga čini lako prenosivim i praktičnim [5]. Prati svih deset prstiju s preciznošću od jedne stotine milimetra te generira i do 200 okvira po sekundi, ovisno o načinu rada uređaja. Naravno, ovisno o aplikaciji u kojoj je uređaj korišten te načinu rada Leap Motion-a, broj okvira može biti znatno manji pošto je ograničen aplikacijom. Vidno polje uređaja je slično obrnuto okrenutoj piramidi čije dvije visine nasuprotnih stranica zatvaraju kut od 150° gdje je raspon očitavanja u dubinu i širinu najviše 60 cm, a u visinu očitava sve od 2.5 do 60 cm [9]. Ovakve specifikacije Leap Motion čine različitim od Kinect-a u tome da je rezolucija Leap Motion-a veća, ali ima manje područje promatranja.

2.2. Kako radi Leap Motion?

Sklopovlje uređaja Leap Motion [9] ima nekoliko dijelova od kojih su glavni dijelovi dva infracrvena senzora, tri infracrvene LED žaruljice te USB kontrolera.

„Mozak“ sklopovlja je USB kontroler koji ima nekoliko funkcija. Senzori i žaruljice mogu se vidjeti na jednom od dijelova Leap Motion-a prikazanim na slici 2. Glavna funkcija je sinkroniziranje slika koje oba senzora šalju, pakiranje sinkroniziranih podataka te slanje tih podataka preko USB priključka u računalo. U osnovnom načinu rada koji balansira broj okvira po sekundi i kvalitetu slika, rezolucija slika koju senzori stvaraju je 640 x 240 slikovnih jedinica koje se generiraju i šalju na računalo brzinom od 115 okvira po sekundi. Osim toga, USB kontroler se brine da se LED žaruljice uključuju samo neposredno prije nego senzori počnu stvarati sliku čime se povećava njihov životni vijek.

Infracrveno zračenje se koristi zato što je našem oku ono nevidljivo pošto početak raspona valne duljine infracrvenog zračenja počinje na kraju raspona valnih duljina koje ljudsko oko može vidjeti [10]. Time se dobije da iako je ruka osvjetljavana te uređaju prikazana kao svijetla, korisnik tu svjetlost neće primijetiti tako da svjetlost LED žaruljica neće smetati korisniku prilikom korištenja. Osim toga, to rezultira i time da uređaj radi dosta dobro i u prostorijama s manjkom svjetla. Pozadina koju Leap Motion vidi (najčešće strop) će tada biti tamna ili barem tamnija od objekta koji senzor prati jer će svjetlost LED žaruljica na udaljenijim predmetima u prostoru očito biti slabija.

Nakon što podaci dođu do računala, primjenjuje se zahtjevna matematika gdje se usporedbom dvodimenzijskih okvira dobivenih iz dva senzora stvaraju podaci koji daju trodimenzijski položaj promatranih objekata.

Slika 2. Sklopovlje uređaja Leap Motion [11]

2.3. Kako koristiti Leap Motion?

Uređaj Leap Motion koristi se tako da se preko USB kabela spaja na računalo te se postavlja između korisnika i ekrana na ravnu površinu tako da je pogled korisnika na ekran monitora okomit na najdužu stranu uređaja. Kako bi bilo jednostavnije odrediti kako okrenuti Leap Motion, u njega je ugrađena indikacijska lampica koja bi trebala biti okrenuta prema korisniku prilikom korištenja uređaja, no čak i u slučaju da se okrene za 180° od željenog položaja, uređaj će sam shvatiti da je okrenut naopako koristeći informaciju o položaju podlaktice korisnika.

Leap Motion će u obliku pop-up notifikacije korisniku dojaviti ukoliko očita bilo kakvu mrlju na strani na kojoj se nalazi senzor, te bi, ukoliko se to dogodi, korisnik što prije trebao očistiti uređaj jer čak i najmanja mrlja može umanjiti iskustvo korištenja uređaja. Uređaj se čisti tako da se prvo iskopča iz računala te tek tada obriše suhom krpom. Osim mrlja, dodatna poteškoća može biti izvor svjetla te zrcalne površine postavljene nasuprot uređaju [12].

Prilikom korištenja, treba pripaziti da između ekrana računala te korisnika ima dovoljno prostora kako bi dijelovi korisnikovog tijela (isključujući ruke), nakit, odjeća ili slično što manje ulazili u fokus uređaja. Korištenje ponekad zna biti dosta zamorno jer se ruke postavljaju u neprirodne položaje tako da je preporučljivo podlaktice ili laktove imati naslonjene na stolu kako bi se ruke što manje umarale. Isto tako treba pripaziti da prije početka korištenja uređaja zglobovi i laktovi ne budu previše savijeni jer to prvi znak da između korisnika i ekrana računala nema dovoljno prostora.

Slika 3. Korištenje Leap Motion-a [13]

2.4. Analiza preciznosti i pouzdanost uređaja Leap Motion

Preciznost i pouzdanost uređaja Leap Motion analizirao je tim inženjera s Fakulteta Elektrotehnike iz Ljubljane [14]. Iz pokusa su htjeli otkriti je li moguće ovaj uređaj koristiti kao zamjenu profesionalnim optičkim sustavima za praćenje pokreta koji imaju veliki broj okvira po sekundi te visoku preciznost. U ovom radu bit će ukratko objašnjena mjerenja koja su bila provedena u spomenutom radu te zaključak koji proizlazi iz dobivenih podataka.

2.4.1. Mjerenja

U pokusima su korištene visoko precizne profesionalne kamere koje su služile kao referenca za usporedbu s podacima dobivenih iz Leap Motion-a. Provedena su dva tipa mjerenja, statičko i dinamičko mjerenje.

Statičko mjerenje provedeno je na način da se na stalak postavio plastični model ruke (Slika 4.a). Na srednji prst ruke je postavljen marker koji je bio referentna točka za mjerenje položaja ruke. Ruka se postavljala u 37 različitih položaja unutar prostora očitavanja Leap Motion-a te su se dobiveni prostorni položaji markera uspoređivali s položajima dobivenih iz profesionalnih kamera. Najmanja standardna devijacija od 0.0081 mm je pronađena na 30 cm točno iznad uređaja dok je najveća

od čak 0.49 mm pronađena u lijevom gornjem kutu od uređaja. Osim toga, primijećene su nestabilnosti i velike varijacije u brzini uzorkovanja između različitih mjerenja gdje standardna devijacija bila 12 Hz.

Dinamičko mjerenje provedeno je na način da se umjesto plastične ruke koristio poseban alat koji je simulirao dva prsta ljudske ruke. Alat je bio napravljen od dva štapića koja su na vrhovima imali pričvršćene markere. Štapići su međusobno bili povezani tako da svojim oblikom tvore slovo „V“ čime se dobilo da je udaljenost između dva markera u svim trenucima približno konstantna (Slika 4.b). Alatom je rukovala osoba tako da je mijenjala njegov položaj unutar vidnog polja Leap Motion kontrolera brzinom od otprilike 10 cm po sekundi. U ovom mjerenju fokus je bio izmjeriti distorziju percepcije prostora Leap Motion kontrolera koja se računala kao standardna devijacija udaljenosti između dva markera. Pošto je udaljenost markera uvijek bila konstanta, ovom metodom se moglo vidjeti kako će uređaj percipirati tu udaljenost s obzirom na položaj u vidnom polju. Iz mjerenja se pokazalo da distribucija standardne devijacije nije normalna (Gaussova krivulja), nasuprot očekivanom. Najveći uzrok tome su veća odstupanja koja se događaju kada se promatrani objekt udaljuje od kontrolera, a posebice kada bi položaj markera prešao udaljenost od 25 cm od uređaja u visinu. Isto kao i u statičkom mjerenju, primijećene su nestabilnosti u brzini uzorkovanja između različitih mjerenja.

a)

b)

Slika 4. a) Statičko mjerenje, b) Dinamičko mjerenje

2.4.2. Zaključak

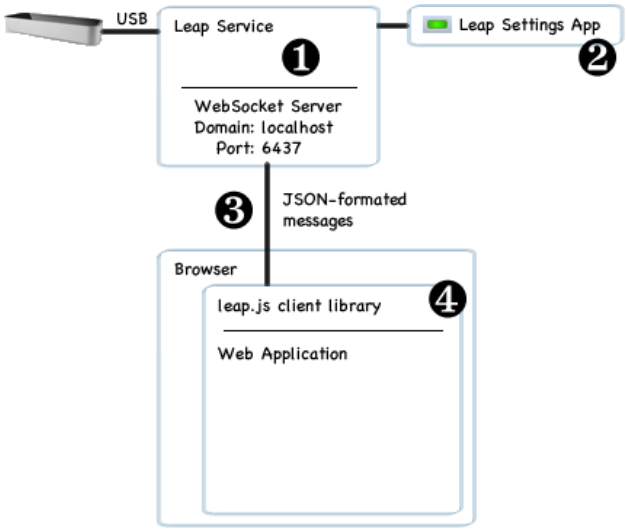
Analiziranjem dobivenih rezultata, procijenjeno je da uređaj ne može biti korišten kao profesionalni uređaj za praćenje pokreta. Tom zaključku najviše pridonosi činjenica da uređaj ima ograničen prostora u kojem senzori daju dovoljno dobre rezultate gdje se ponekad iz neobjašnjivih razloga dobivaju bolji ili lošiji rezultati s obzirom na položaj ruke u vidnom polju uređaja. Najtočnija očitavanja se dobivaju kada marker prelazi preko uređaja jer je tada cijeli objekt najbolje vidljiv. Druga važna činjenica koja pridonosi danom zaključku je nekonzistentnost frekvencije uzorkovanja i u statičnom i u dinamičnom mjerenju.

Iako možda ne može biti korišten za profesionalnu upotrebu, Leap Motion predstavlja revolucionarni uređaj za praćenje pokreta koji se može vrlo dobro iskoristiti u računalnim aplikacijama i simulacijama.

3. Leap Motion aplikacijsko programsko sučelje

Aplikacije koje koriste Leap Motion, dobivaju podatke od Leap Motion kontrolera preko Leap Motion usluge instalirane na korisnikovu računalu. Programska oprema (SDK) za razvoj Leap Motion aplikacija nudi dvije vrste API-a (aplikacijsko programskog sučelja), osnovno sučelje i sučelje preko web priključnica [15]. Za prikaz mogućnosti Leap Motion-a u ovom radu, funkcionalnosti će se razvijati nad web aplikacijom, stoga će nadalje biti fokus na sučelje preko web priključnica (engl. web socket), a korišten programski jezik bit će Javascript.

Sučelje preko web priključnica radi tako da na lokalnoj adresi računala, na vratima (engl. port) 6437 pokrene server koji koristi web priključnice. Sučelje šalje podatke koje prima od kontrolera u obliku JSON [36] poruka. Podaci se primaju i obrađuju koristeći biblioteku leap.js [16] koja uspostavlja konekciju prema pokrenutom serveru, prima JSON poruke s podacima (Slika 5.) te je šalje web aplikaciji koja je koristi.

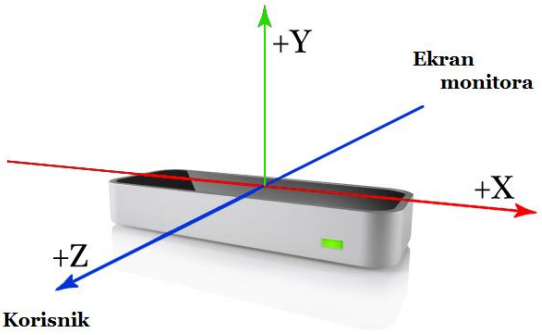


Slika 5. Rad Leap Motion sučelja preko web priključnica [15]

3.1. Koordinatni sustav

Leap Motion API koristi desni Kartezijev koordinatni sustav. Ishodište je točka na vrhu te u sredini Leap Motion kontrolera. Os *x* i os *z* tvore horizontalnu ravninu. Os *x* prostire po dužem dijelu uređaja Leap Motion-a (okomito na korisnikov pogled) gdje je negativna vrijednost *x* osi lijevo od ishodišta, a pozitivna desno. Os *z* se prostire po kraćem dijelu uređaja Leap Motion (u smjeru korisnikova pogleda) s time

da je pozitivni dio *z* osi prostor od ishodišta prema korisniku, a negativni od ishodišta prema ekranu korisnika (pretpostavljajući da je uređaj postavljen tako da je između ekrana i korisnika). Os *y* je vertikalna komponenta sustava koja uvijek ima pozitivne vrijednosti (Slika 6.). Vrijednosti na osima su udaljenosti mjerene u milimetrima.



Slika 6. Leap Motion koordinatni sustav [17]

3.2. Osnovni razredi biblioteke leap.js

3.2.1. Razred Controller

Razred *Controller* (hrv. kontroler) je glavno sučelje prema Leap Motion kontroleru. Njegova glavna zadaća je tijekom svake iteracije primiti novi okvir s uređaja te obraditi taj okvir onako kako korisnik razreda to zahtjeva. Osim toga, razred ima pristup konfiguracijskim postavkama. *Controller* sprema povijest do 60 okvira.

Postoji dva načina kako napraviti novu instancu razreda *Controller*. Prvi način je pozivom metode *Leap.loop()* koja prima dva argumenta te vraća već inicijalizirani i pokrenuti kontroler. Prvi argument, koji nije nužan, su opcije koje želimo postaviti. Neke od opcija su broja vrata (engl. port) na kojem želimo da pokrenuti web server sluša te mogućnost praćenja gesti. Drugi argument je funkcija koja će se pozivati svaki puta kada se naš internet preglednik sprema iscrtavati na ekran. Funkcija ima jedan argument, frame (hrv. okvir) koji predstavlja trenutno najsvježiji okvir. Programski kod 1. prikazuje stvaranja nove instance *Controller* koristeći *Leap.loop* metodu.

```
//enableGestures je inicijalno postavljen na false
//postavljanjem na true se omogućuje prepoznavanje gesti
var controllerOptions = {enableGestures: true};

var newController = Leap.loop(controllerOptions, function(frame) {
    //tijelo funkcije koje se poziva svakim
    //osvježavanjem ekrana internet preglednika
});
```

Programski kod 1. Kreiranje instance *Controller* razreda *Leap.loop* metodom

Drugi način stvaranja instance *Controller*-a je pozivom konstruktora *new Leap.Controller()* koji kao argument može primiti opcije (Programski kod 2.). Razlika ovog i prijašnjeg načina je ta da *Leap.loop()* pokreće praćenje kontrolera, dok ovim načinom korisnik izabire kada počinje praćenje obrada okvira pozivom metode *connect()* nad instancom *Controller*-a. Kao što se može primijetiti, prilikom inicijalizacije Controller instance korištenjem drugog načina, ne odabiremo funkciju koja će se izvršavati na svaki okvir. Okvir se može obrađivati tako da u petlji koju smo sami stvorili svaki put tražimo novi okvir pozivom metode *frame()* nad instancom *Controller*-a ili toj instanci odredimo kako da se ponaša na određene događaj tipa *„frame“* čime nam se petlja stvara sama. Način na koji se instanci *Controller*-a odredi kako da se ponaša na ostvarivanje nekog od događaja je pozivom metode *on()* koja prima dva argumenta. Prvi argument je ime događaja što može biti primjerice „frame“ ili „gesture“ (hrv. gesta). Drugi argument je funkcija koja će se pozvati kada se ostvari željeni događaj. Tako će se funkcija za događaj „frame“ pokrenuti svaki puta kada svakim osvježavanjem ekrana korisnika, a događaj „gesture“ svaki puta kada se unutar okvira prepozna neka od gesti.

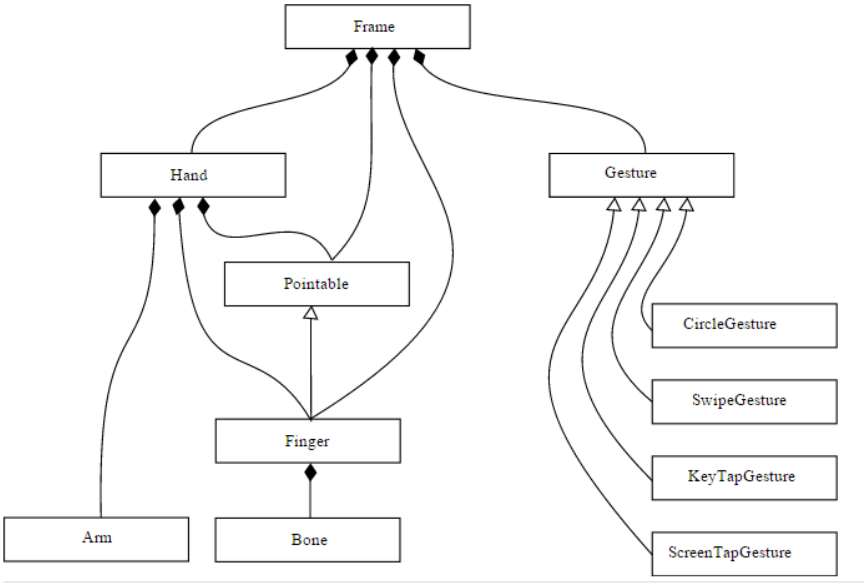
Od važnijih metoda koju posjeduje instanca *Controller*-a, bitno je spomenuti metodu *use()* koja se koristi kako bi se na *Controller*-u dojavilo korištenje dodatka (engl. plug-in). Više o tome u poglavlju 3.4. ovoga rada.

```
var controllerLeap = new Leap.Controller();

//kako će se obrađivati svaki okvir
controllerLeap.on("frame", function (frame) {
    //obradi okvir
});
//započni praćenje okvira
controllerLeap.connect();
```

Programski kod 2. Kreiranje instance *Controller* razreda *new Leap.Controller()* konstruktorom

3.2.2. Razred Frame

Razred Frame (hrv. okvir) je „korijen“ svih podataka s Leap Motion kontrolera koje su na raspolaganju korisniku. *Frame* daje pristup svim objektima koje se prate od strane kontrolera. Nova instanca *Frame* razreda stvara se svaki put kada se obavlja osvježavanje ekrana internet preglednika. Dio informacija koju sadrži jedna *Frame* instanca prikazan je na Slici 7.



Slika 7. Sadržaj *Frame* instance [18]

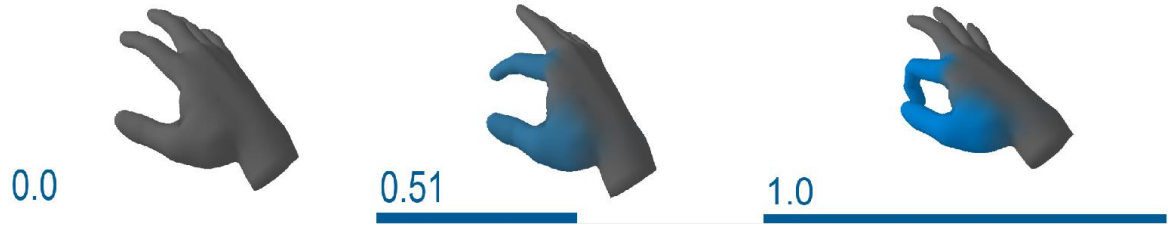
Frame instanca sadrži nizove svih ruku, prstiju, predmeta koji se prate te gesti. Kako bi se tijekom više okvira mogla pratiti promjena položaja ruke, prsta ili stanja geste, instanca *Frame* razreda nudi metode kojima uz predani *id* objekta, dobivamo traženi objekt, ako takav postoji u trenutnom okviru. Identifikacijska oznaka svakog prepoznatog objekta će se čuvati od kada je objekt ušao do kada ne izađe iz vidnog polja kontrolera što znači da će se svaki okvir objekti kreirati nanovo, no ako su prepoznati i kao dio prijašnjih okvira, postavit će im se ista identifikacijska oznaka.

3.2.3. Razred Hand

Ruke su glavni predmet promatranja za Leap Motion kontroler. Kontroler u sebi čuva model ruke te uspoređuje taj model sa slikom koju dobiva preko senzora kako bi prepoznao ljudsku ruku. To mu omogućuje da radi pretpostavke gdje se neki od prstiju nalazi, čak i kada prst nije senzoru vidljiv. Razred *Hand* (engl. ruka) služi kao reprezentacija ljudske ruke koja se prati preko Leap Motion kontrolera. U nastavku slijede neke važni atributi i metode razreda *Hand*:

- *fingers* – niz instanci razreda *Finger* trenutno vidljivih u okviru.
- *finger(id)* – metoda koja vraća instancu razreda *Finger* ako postoji instanca tog razreda koji ima tu identifikacijsku oznaku te dio je ruke nad kojom se metoda poziva.
- *confidence* – broj između 0 i 1 koji nam govori koliko dobro unutarnji model ruke odgovara snimljenim podacima.
- *pinchStrength* - broj između 0 i 1 koji nam govori koliko je jaka poza prstohvata (Slika 8.).
- *grabStrength* - broj između 0 i 1 koji nam govori koliko je jaka poza stiska šake (Slika 9.).
- *type* – tip ruke, ako je ruka lijeva, vrijednost je „left“, ako je ruka desna, vrijednost je „right“.

Osim spomenutih, tu su i atributi za svaki od prstiju ruke zasebno nazvani po njihovim engleskim imenima. Ruka isto tako sadrži položaj dlana, normalu dlana, smjer ruke koji se gleda od položaja dlana prema prstima, itd. Programski kod 3. pokazuje obavljanje određene akcije nad instancom razreda *Hand* ako je ruka desna te ako joj je snaga stiska šake veća od 0.9.



Slika 8. Snaga prstohvata [19]

Slika 9. Snaga stiska šake [20]

```
controllerLeap.on("frame", function (frame) {
    var hands = frame.hands;

    //ako unutar vidnog polja senzora nema niti jedne ruke, završi
    if(hands.length === 0) {
        return;
    }

    for(var i = 0; i < hands.length; i++) {
        var hand = hands[i];
        if(hand.type === "right" && hand.grabStrength > 0.9) {
            //napravi akciju
        }
    }
});
```

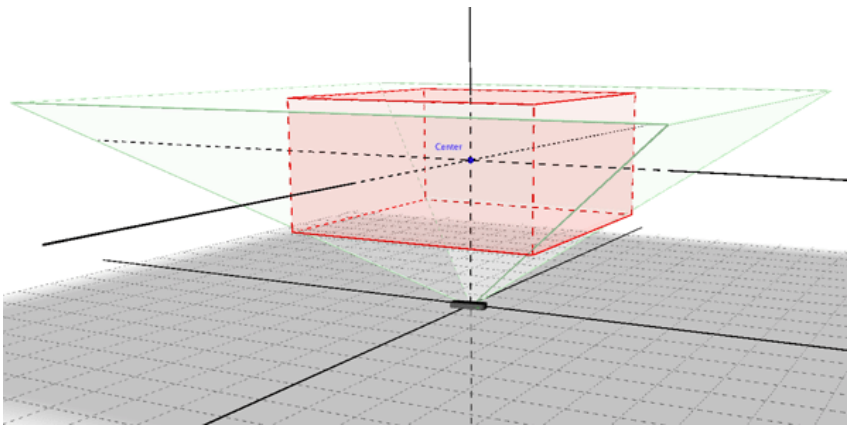
Programski kod 3. Provjera vrijednosti atributa instance razreda *Hand*

3.2.4. Razred Finger

Razred *Finger* (hrv. prst) služi kao reprezentacija ljudskog prsta. Uređaj Leap Motion očitava poziciju svake kosti prsta te se položaju i usmjerenju svake od njih može pristupati posebno. Atribut *type* daje informaciju kojeg tipa je prst. Vrijednosti tipova su brojevi od 0 do 4 s time da 0 predstavlja palac, 1 kažiprst, 2 srednji prst, 3 prstenjak, te 4 mali prst.

3.2.5. Razred Interaction Box

Razred *InteractionBox* (hrv. interakcijska kutija) je područje u obliku kutije (kvadar) koje se kompletno nalazi unutar vidnog polja Leap Motion kontrolera (Slika 10.). *InteractionBox* služi kako bi točku iz stvarnog prostora normalizirali na vrijednosti od 0 do 1 za svaku od tri dimenzije. Ishodište interakcijske kutije je lijevi, donji i bliži kut gledajući od korisnika prema uređaju Leap Motion. Instancu razreda dobivamo kao atribut *interactionBox* instance razreda *Frame*.



Slika 10. Interakcijska kutija

3.3. Geste

Geste su prepoznatljivi pokreti ruku. Uređaj Leap Motion prati pokrete ruke te klasificira određene pokrete kao geste. Kada uređaj prepozna gestu, dodjeljuje joj identifikacijsku oznaku te je stavlja u niz gesti nove instance razreda *Frame*. Razred koji predstavlja geste zove se *Gesture* (hrv. gesta). *Gesture* razred ima nekoliko važnih atributa:

- *id* – identifikacijska oznaka geste. Ovaj atribut je važan jer se trajanje jedne geste može prostirati kroz više različitih okvira.
- *handIds* – niz identifikacijskih oznaka ruku koje su sudjelovale u gesti.
- *pointableIds* – niz identifikacijskih oznaka objekata kojima je moguće pokazivati (prst ili predmet) koji su sudjelovali u gesti
- *state* – trenutno stanje u kojem se gesta nalazi. Postoje tri stanja u kojem se gesta može nalaziti: „start“, „update“ i „stop“. Neke od gesti mogu biti u sva tri stanja, dok neke mogu biti samo u stanje „stop“ što znači da se takva gesta ne prostire kroz više različitih okvira, nego je sadržana samo u tom jednom.
- *type* – tip geste. Trenutni podržani tipovi gesti unutar leap.js biblioteke su: „keyTap“, „screenTap“, „circle“ i „swipe“.

Iako postoji samo četiri različita tipa gesti, bitno je napomenuti da geste nikada ne obavlja sama ruka, već je to uvijek neki od objekata kojima je moguće pokazivati (prst ili predmet). To znači da se postoji puno mogućnosti za korištenje spomenutih gesti, primjerice na način da akcije ovise o tipu prsta koji gestu obavlja ili o smjeru kojim se gesta prostire. Programski kod 4. prikazuje primjer prepoznavanja gesti te ispisivanja tipa geste u konzolu.

```
//omogućavanje gesti postavljanjem enableGestures na true
var controller = Leap.loop({enableGestures: true}, function(frame){

    //ako je neka od gesti prepoznata
    if(frame.gestures.length > 0){
        frame.gestures.forEach(function(gesture){
            switch (gesture.type){
                case "circle":
                    console.log("Circle Gesture");
                    break;
                case "keyTap":
                    console.log("Key Tap Gesture");
                    break;
                case "screenTap":
                    console.log("Screen Tap Gesture");
                    break;
                case "swipe":
                    console.log("Swipe Gesture");
                    break;
            }
        });
    }
});
```

Programski kod 4. Prepoznavanje tipa gesti

3.3.1. Gesta KeyTap

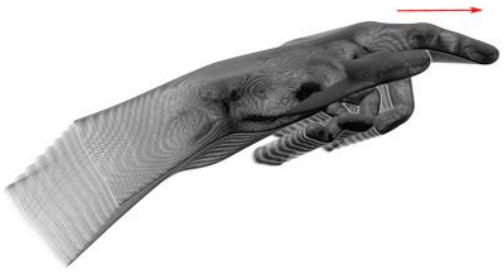
Gesta *KeyTap* (hrv. dodir tipke) je gesta koja se obavlja tako da korisnik napravi pokret prsta slično kao da pokušavamo nešto snažnije pritisnuti tipku na tipkovnici (Slika 11.). Potrebno je obaviti pokret prstom prema dolje, te vratiti prst u položaj blizu početnoga. Pokreti ruke bi za vrijeme izvođenja ove geste trebali biti minimalni. Pojedina *KeyTap* gesta će uvijek biti sadržana samo u jednom okviru što znači da će njezin atribut *state* uvijek imati vrijednost „stop“.

Slika 11. Gesta *KeyTap* [22]

3.3.2. Gesta ScreenTap

Gesta *ScreenTap* (hrv. dodir ekrana) je gesta koja se obavlja tako da korisnik u kratkom periodu pomakne ruku prema naprijed te je vrati u približno isti položaj

kao i na početku, imajući pri tome jednim ispruženi prst (Slika 12.). Pokret je sličan pokretu koji se obavlja prilikom korištenja dodirne podloge. Pojedina *ScreenTap* gesta će uvijek biti sadržana samo u jednom okviru što znači da će njezin atribut *state* uvijek imati vrijednost „stop“.

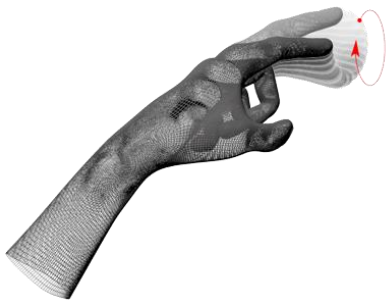


Slika 12. Gesta *ScreenTap* [22]

3.3.3. Gesta Circle

Gesta *Circle* (hrv. kružiti) je gesta koja se obavlja tako da korisnik pokretom prsta napravi oblik sličan krugu (Slika 13.). Atribut *state* poprimit će vrijednost „start“ kada je pokret dovoljno dug da bi ga se moglo klasificirati kao krug, vrijednost „update“ kako se stvaranje kruga nastavlja te vrijednost „stop“ kada je cijeli pokret dovršen. Osim spomenutih atributa nadrazreda *Gesture*, razred *Circle* ima još neke korisne atribute. To su:

- *center* – centar kruga stvorenog pokretom.
- *normal* – normala kruga koja služi kako bi se odredilo je li pokret napravljen u smjeru kazaljke na satu ili obrnuto.
- *progress* – vrijednost koja govori koliko puta je prst obišao krug. Tako će vrijednost 0.5 značiti da je prst obišao pola kruga, dok će vrijednost 5 značiti da je prst pet puta obišao krug.
- *radius* – radius kruga.



Slika 13. Gesta *Circle* [22]

3.3.4. Gesta Swipe

Gesta *Swipe* (hrv. udariti) je gesta koja se obavlja tako da korisnik pomiče cijelu ruku u širinu ili visinu tako da normala dlana ruke bude približno u smjeru ili obrnuto od smjera kretanja ruke (Slika 14.). Pokret za ovu gestu je sličan kao da korisnik ispred sebe ima loptu koju želi odgurnuti u stranu svojim dlanom ili stražnjim dijelom ruke. Atribut *state* poprimit će vrijednost „start“ pri početku pokreta, vrijednost „update“ dok pokret traje te vrijednost „stop“ kada je cijeli pokret dovršen. Osim spomenutih atributa nadrazreda *Gesture*, razred *Swipe* ima još neke korisne atribute. To su:

- *direction* – jedinični vektor smjera pokreta.
- *speed* – brzina prsta koji izvodi pokret.
- *startPosition* – početna pozicija pokreta.
- *position* – trenutna pozicija pokreta.

Slika 14. Gesta *Circle* [22]

3.4. Leap Motion API dodaci (engl. plug-ins)

Biblioteka leap-plugins.js [23] sadrži najvažnije dodatke biblioteci leap.js kojima se korisnik može poslužiti u razvijanju Leap Motion web aplikacije. Kako bi se biblioteka mogla koristiti, potrebno ju je dodati u HTML dokument u kojem se nalazi aplikacija. Dodatak će biti korišten samo kada se nad *Controller* instancom pozove metoda *use* koja prima dva argumenta. Prvi argument je ime dodatka koji

se koristi, a drugi, opcionalni, su parametri konfiguracije dodatka kojim korisnik određuje na koji će način koristiti taj dodatak. Dodavanjem dodataka u instancu *Controller,* ona dobiva novi atribut naziva istog kao ime dodatka gdje je moguće pristupiti određenim podacima dodanog dodatka. Primjer pristupa dodatku „dodatakX“ dodanom u instancu *Controller* nazvanu *leapController* bio bi *leapController.plugins.dodatakX.* Neki dodaci nisu dio biblioteke leap-plugins.js, nego su samostalne biblioteke.

Dodatak *handEntry* okida događaj na *Controlleru* kada ruka uđe u vidno polje uređaja te kada iz njega izađe.

Dodatak *screenPosition* dodaje metodu *screenPosition* rukama, prstima i prepoznatljivim predmetima koja vraća položaj objekta na ekranu.

Dodatak *transform* transformira položaj ruku u odnosu na njezinog objekta roditelja. Ruke se mogu translatirati, rotirati ili skalirati kroz opcije. Atribut opcija *effectiveParent* postavljaju predani objekt kao objekt roditelj rukama u aplikaciji. Ukoliko taj atribut nije postavljen, objekt roditelj je scena. Programski kod 5. prikazuje namještanje dodatka *transform* instanci *Controller*.

Dodatak *riggedHand* [24] se za razliku od prijašnjih dodataka ne nalazi u leap-plugins.js biblioteci, nego u leap-rigged-hand.js biblioteci. Ovaj dodatak omogućuje prikaz modela ruku kao objekta scene. Slika 8. pokazuje izgled modela ruku kojeg nudi ovaj dodatak.

```
controllerLeap = new Leap.Controller({
    enableGestures: true
}).use('transform', { //opcije
        effectiveParent: camera
});
```

Programski kod 5. Postavljanje dodatka *transform*

4. Micro Soot maintenance app (MSMA)

U ovom radu, objasnit će se ugrađivanje Leap Motion funkcionalnosti u web aplikaciju. Web aplikacija koja će biti korištena u tu svrhu je aplikacija koja simulira postupak održavanja uređaja Micro Soot, naziva Micro Soot maintenance app i sadrži nekoliko različitih načina rada. Aplikaciju su kroz svoje diplomske radove razvili Ana Nekić [25] i Marko Pilipović [26]. U ovom poglavlju bit će objašnjen uređaj Micro Soot, čiji je model korišten u aplikaciji, funkcionalnosti aplikacije te korištene biblioteke koje su se koristile kako bi se te funkcionalnosti mogle ostvariti.

4.2. Uređaj Micro Soot

Uređaj Micro Soot (Slika 15.) napravljen je od strane tvrtke AVL [27]. Micro Soot je sustav za kontinuirano mjerenje koncentracije čađe u ispušnim plinovima dobivenih iz motora s unutarnjim izgaranjem. Uređaj ima mogućnost mjerenja čađi bez da na rezultat utječu ostali elemente unutar ispušnog plina. Isto tako jedna od važnijih karakteristika mu je visoka osjetljivost s rezolucijom od 1 µg/m³. Pošto je uređaj vrlo osjetljiv [28] te se prema njemu mora pristupati s velikom pažnjom, rukovanje njime pripada samo kvalificiranom osoblju. To ponekad predstavlja problem jer je osposobljavanje novog osoblja skupo, a uređaj je potrebno često pregledavati i održavati. Kako bi se olakšao postupak održavanja uređaja čak i manje kvalificiranim osobama, napravljena je web aplikacija kojoj je jedna od funkcionalnosti da sadrži način rada u kojem je taj postupak detaljno objašnjen koristeći nove tehnologije prikazivanja 3D grafike na web-u. Opis funkcionalnosti aplikacije te način na koje su te funkcionalnosti implementirane opisan je u nastavku.

Slika 15. Uređaj Micro Soot [26]

4.3. Načini rada aplikacije MSMA

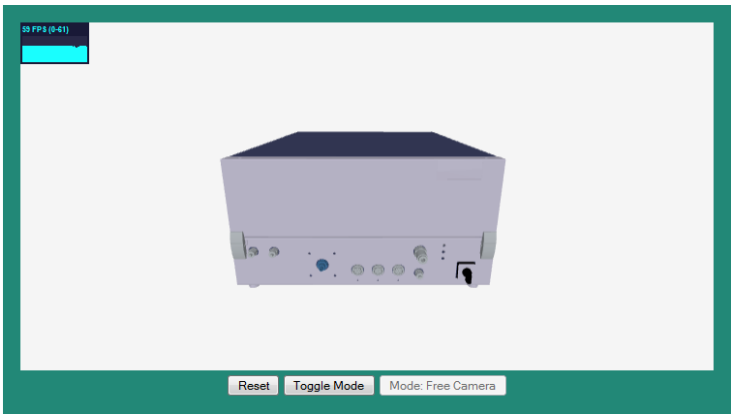
Aplikacijom se upravlja računalnim mišem ili dodirnom podlogom (engl. Touchpad), no u opisu funkcionalnosti fokus će biti na upravljanje računalnim mišem. U aplikaciju je na početku učitan Micro Soot model kao osnovni objekt. Unutar aplikacije postoji 5 različitih načina rada, svaki sa svojim funkcionalnostima. Neki načini rada međusobno dijele dio funkcionalnosti, dok su neki međusobno potpuno nezavisni.

Unutar aplikacije postoji glavni izbornik koji nudi mogućnost promjene načina rada (opcija „Toggle Mode“), resetiranje aplikacije u početno stanje (opcija „Reset“) te se osim toga prikazuje naziv trenutnog načina rada. Ovisno o načinu rada, glavni izbornik dodaje ili izbacuje određene opcije.

4.3.1. Način rada „Free Camera“

Početni način rada nazvan „Free Camera“ (hrv. „Slobodna Kamera“) omogućava korisniku da upravlja kamerom 3D scene. Točka pogleda kamere je u početnom stanju usmjerena u središte scene, gdje se nalazi Micro Soot te je namještena tako da je uređaj promatran s njegove prednje strane. Pozicionirana je na takvoj udaljenosti da niti jedan dio objekta ne izlazi iz kadra (Slika 16.). Korisnik u ovom načinu rada ima na raspolaganju 3 različite funkcionalnosti:

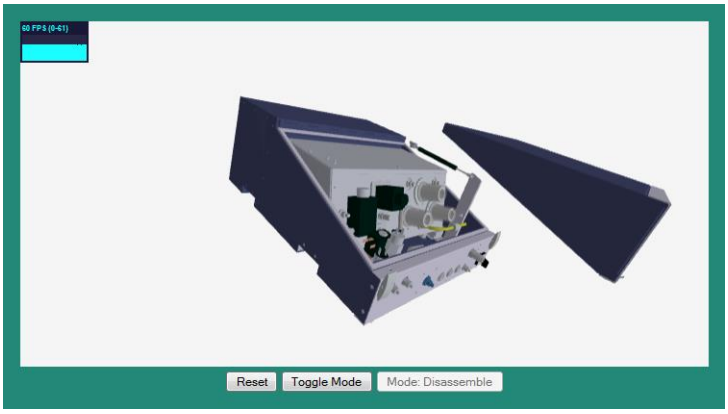
- rotacija kamere oko točke pogleda. Akcija se okida pritiskom lijeve tipke miša te pomicanjem miša u jednu od strana.
- pomicanje kamere tako da se kamera pomiče u stranu mijenjajući pritom svoj položaj te za isti vektor i svoju točku pogleda (engleski naziv takve akcije je „Pan“). Akcija se okida pritiskom desne tipke miša te pomicanjem miša u jednu od strana.
- promjena udaljenosti kamere od njene točke pogleda što uključuje približavanje (engl. Zoom in) i udaljavanje (engl. Zoom out). Akcija se okida pritiskom srednje tipke miša te pomicanjem mišem u jednu od strana ili rotacijom „kotačića“ koji se nalazi na mjestu srednje tipke.



Slika 16. Početno stanje aplikacije s uključenim načinom rada „Free Camera“

4.3.2. Način rada „Disassemble“

U načinu rada „Disassemble“ (hrv. „Rastaviti“) korisnik ima iste funkcionalnosti kao i u „Free Camera“ načinu rada, samo je uloga lijeve tipke miša nešto izmijenjena. Model Micro Soot se sastoji od 39 različitih dijelova, iako djeluje kao kompaktno tijelo. Korisnik u ovom načinu rada ima mogućnost odabrati jedan od dijelova modela te ga pomicati u stranu. Ta akcija se okida kada korisnik pritisne lijevu tipku miša ako i samo ako pokazivač u tom trenutku pokazuje na dio Micro Soot modela. Korisnik tada može pomicati objekt po širini i visini prozora u kojem se nalazi aplikacija. Naravno, pošto se koristi računalni miš, pomicanje objekta u dubinu bi bilo teže izvedivo zbog ograničenosti uređaja. Korisnik još uvijek može koristiti funkcionalnost rotacije načina rada „Free Camera“, unatoč tomu što obje akcije koriste lijevu tipku miša. Ta akcija se okida kada prilikom pritiska lijeve tipke miša, pokazivač ne pokazuje na dio Micro Soot modela. Na Slici 17. je primjer korištenja aplikacije u ovom načinu rada, kamera je zarotirana te je nakon toga jedan od dijelova modela pomaknut u desno.



Slika 17. Način rada „Disassemble“

4.2.3. Način rada „Explode“

U načinu rada „Explode“ (hrv. „Eksplodirati“) korisnik ima iste funkcionalnosti kao i u „Free Camera“ načinu rada, uz neke dodatne funkcionalnosti. Korisniku je u glavnom izborniku aplikacije dodan padajući izbornik gdje može birati jedan od 4 načina eksplozije modela te povratak modela u početno stanje. Eksplozije modela su izvedene tako da se, ovisno o odabranoj eksploziji, neki od dijelova modela pomiču od svog originalnog položaja u drugi predefinirani položaj. Svaka od eksplozija je smisleno napravljena tako da u prvom planu ističe neku od cjelina Micro Soot modela. Opcija „No Explode“ u padajućem izborniku vraća sve dijelove modela u početno stanje. Na Slici 18. prikazana je jedna od eksplozija sa zarotiranom kamerom.

Slika 18. Način rada „Explode“, opcija „Explode0“

4.2.4. Način rada „Teach Me“

U načinu rada „Teach Me“ (hrv. „Nauči Me“) upravljanje kamerom nije omogućeno. Dodana je nova opcija „Run“ u glavni izbornik aplikacije. Korisnik klikom miša na tu opciju dobiva novi set opcija, te se iz glavnog izbornika uklanjaju opcije za resetiranje aplikacije i za promjenu načina rada. Korisnik se može vratiti u prijašnji izbornik klikom miša na opciju „Exit“. Ostale dodane opcije služe za pokretanje određenih animacija modela. Ponuđene opcije su:

- „Prev“ koja pokreće prijašnju animaciju.
- „Play“ koja pokreće trenutnu animaciju.
- „Next“ koja pokreće sljedeću animaciju.

Prilikom pokretanja animacija, ispod prozora aplikacije ukratko se opisuje radnja koja se obavlja. Ovaj način rada služi kako bi korisnika naučio postupak održavanja

uređaja Micro Soot. Postupak je definiran kao niz slijednih akcija koje korisnik mora obaviti.

4.2.5. Način rada „Test Me“

Kao i u načinu rada „Teach Me“, u načinu rada „Test Me“ (hrv. „Ispitaj me“) upravljanje kamerom nije omogućeno. U glavni izbornik dodana je opcija „Run“. Klikom miša na tu opciju korisnik dobiva novi set opcija, te se iz glavnog izbornika uklanjaju opcije za resetiranje aplikacije i za promjenu načina rada. Korisnik se može vratiti u prijašnji izbornik klikom miša na opciju „Exit“. Ponuđena je još jedna dodatna opcija „Help“ koja korisniku ispod prozora aplikacije ispisuje što mu je činiti kako bi prešao u sljedeće stanje ovog načina rada. Cilj ovog načina rada je omogućiti korisniku da ispita svoje znanje o postupku održavanja uređaja Micro Soot kojeg je mogao naučiti u načinu rada „Teach Me“. Korisnik lijevim klikom miša odabire dio Micro Soot modela nad kojim se sljedeća akcija treba izvršiti. Ukoliko je korisnik odabrao ispravni dio modela, pokreće se animacija te se prelazi u novo stanje gdje je korisnikova uloga odabrati sljedeći dio modela i tako sve dok nije došao do kraja postupka. Ukoliko korisnik nije odabrao ispravni dio modela, neće se dogoditi ništa.

4.4. Implementacija

Aplikacija je napravljena kao web stranica na način da se u HTML (Hyper Text Markup Language) [29] dokument ubacila Javascript [30] skripta aplikacije zajedno sa svim Javascript bibliotekama potrebnim za uspješno izvršavanje te skripte. U nastavku bit će ukratko objašnjene tehnologije koje omogućuju nastanak ovakve aplikacije te korištene Javascript biblioteke.

4.4.1. HTML5 i canvas

HTML5 [31] je trenutno posljednja verzija standarda prezentacijskog jezika za izradu stranica HTML. Iako je HTML5 unio puno različitih novina, u ovoj web aplikaciji je bitna samo jedna od njih, a to je element s oznakom „canvas“ ili na hrvatskom - platno. Element samo po sebi ne crta ništa, ali je spremnik za grafiku koju crtamo upotrebom skripti (obično JavaScript jezika) i tako omogućava crtanje. Crtanje se izvršava tako da se puni kontekst platna koji je samostalno dovoljan kada

se radi grafika u dvije dimenzije, a uz pomoć WebGL-a kada je potrebna i treća dimenzija. MSMA koristi 3D grafiku, stoga koristi WebGL.

4.4.2. WebGL

WebGL (Web Graphics Library) [32] je Javascript API (aplikacijsko programsko sučelje) za ostvarivanje prikaza interaktivne 3D i 2D grafike. WebGL to čini tako da pruža sučelje koje prilagođava OpenGL ES [33] tako da ga se može koristiti unutar HTML5 elementa platno. Prednost WebGL-a je što može iskoristiti sklopovsko ubrzanje koje djeluje na način da akcije za koje je grafička procesorska jedinica (GPU) prikladnija i koje može obaviti jako brzo, obavi GPU čime se središnja procesorska jedinica (CPU) rasterećuje i omogućuje joj se obavljanje drugoga posla. WebGL je neovisan je o platformi i podržavaju ga svi najveći internetski preglednici [34] bez instalacije dodataka (engl. plug-ins). Jedna od poteškoća u korištenju WebGL-a je da se prilikom programiranja mora razmišljati na dosta niskoj razini što uključuje dobro poznavanje matematike i pisanje programa za sjenčanje (engl. shaders).

4.4.3. Three.js

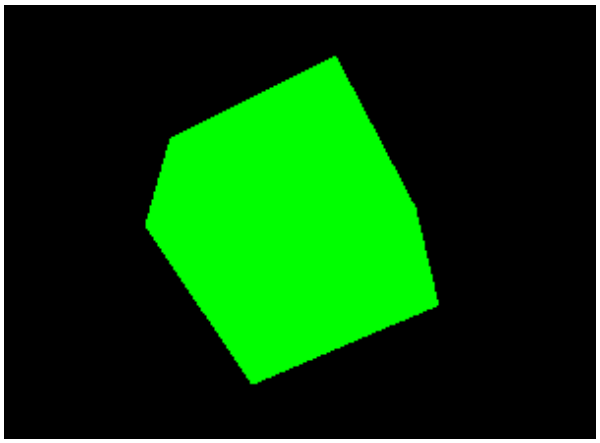
MSMA koristi WebGL, ali na indirektan način, preko omotača oko WebGL API-a, biblioteke Three.js. Three.js biblioteka omogućava rad sa grafikom na višoj razini od WebGL, objektno je orijentirana te skriva složenu matematiku od korisnika, iako u isto vrijeme daje mogućnost korisniku da koristi i matematički zahtjevnije dijelove, ako mu je potrebno [35]. Osnovni primjer korištenja Three.js biblioteke dan je u nastavku pod oznakom „Programski kod 6.“. Sličnu inicijalizaciju ima i MSMA samo je nešto proširena.

U ovom isječku koda se pozivom metode *requestAnimationFrame* kroz parametar metode na svaki novi okvir internet pregledniku javlja da pozove funkciju *render.* Pošto funkcija poziva samu sebe, ovaj poziv će se beskonačno dugo izvršavati. Ideja metode je da na svaki okvir javi osvježivaču prikaza da osvježi prikaz predavši mu pritom scenu koju treba prikazati i kameru s koje će korisnik promatrati scenu. Za inicijalizaciju kamere potrebno je definirati redom: vidno polje, odnos širine i visine prozora u kojem će scena biti iscrtana, udaljenost do najbliže ravnine odsijecanja te udaljenost do najdalje ravnine odsijecanja. Osvježivaču prikaza (engl. renderer) se postavlja širina i visina HTML elementa platno koje će stvoriti. U ovom

slučaju, platno je veličine cijelog prozora, no u slučaju aplikacije MSMA, platno će biti smješteno unutar jednog od *div* elemenata u html dokumentu. Takav element nazivamo kontejnerom. Kako scena ne bi bila prazna, u ovom primjeru se stvara kocka tako da se definira nova rešetka (engl. mesh) s već prije definiranom geometrijom i materijalom koji je u ovom slučaju samo zelena boja. Kocka je tada dodana u scenu pozivom metode *add*. Prije samoga početka kamera se postavlja na dovoljnu udaljenost kako bi se promatrani objekt mogao uočiti. Na svaki okvir kocka se rotira tako da joj se mijenjaju *x* i *y* vrijednosti Eulerovih kutova rotacije zadanim u radijanima.

```
<html>
    <head>
        <title>My first Three.js app</title>
    </head>
    <body>
        <script type="text/javascript" src="js/three.js"></script>
        <script>
            var scene = new THREE.Scene(); //stvaranje scene
            var camera = new THREE.PerspectiveCamera(75,
                    window.innerWidth / window.innerHeight,
                    0.1, 1000); //stvaranje kamere

            //stvaranje osvježivača prikaza
            var renderer = new THREE.WebGLRenderer();
            renderer.setSize(window.innerWidth, window.innerHeight);
            document.body.appendChild(renderer.domElement);

            //stvaranje geometrije za kocku i materijala (zelena)
            var geometry = new THREE.BoxGeometry(1, 1, 1);
            var material = new THREE.MeshBasicMaterial({color: 0x00ff00});
            var cube = new THREE.Mesh(geometry, material);
            scene.add(cube); //dodavanje kocke u scenu

            camera.position.z = 5; //postavljanje pozicije kamere

            var render = function () {
                //na svaki okvir pozovi funkciju render
                requestAnimationFrame(render);

                // animacija kocke
                cube.rotation.x += 0.1;
                cube.rotation.y += 0.1;

                //iscrtavanje promjena
                renderer.render(scene, camera);
            };

            render();
        </script>
    </body>
</html>
```

Programski kod 6. Početni Three.js primjer



Slika 19. Prikaz generiran primjerom „Programski kod 6.“

Ovaj primjer, iako kratak, pokazuje jednostavnost obavljanja transformacija nad objektima u sceni. Aplikacija MSMA na isti način obavlja osvježavanje prikaza, samo što dodaje još neke dodatne elemente u scenu. Neki od tih objekata su usmjerena i okolna svjetla. Usmjerena svjetla djeluju nad cijelom scenom iz nekog smjera, u aplikacijama se najčešće koriste kao izvor svjetlosti koje predstavlja sunce. Okolna svjetla su primijenjena na sve objekte u sceni globalno. Osim objekata svjetla, isto tako se učitava i Micro Soot model tako da se učitava svaki od njegovih dijelova posebno. Dijelovi modela su zapisani kao JSON [36] objekti te se učitavaju pomoću *JSONLoader* razreda Three.js biblioteke.

Razred *Raycaster* (hrv. Odašiljač zraka) omogućava slanje zrake koja počinje u nekoj poziciji te ide u smjeru definiranim vektorom. Pozicija i vektor smjera su predani kao parametri konstruktora. Ta razred se u aplikaciji MSMA koristi u dva načina rada, „Disassemble“ i „Test Me“. U oba načina rada služi za to da bi se zraka, koju šalje odašiljač zraka, poslala od pozicije pritiska tipke računalnog miša u smjeru u kojem gleda kamera. Tada se pozivom metode *intersectObject* nad odašiljačem zraka provjerava siječe li zraka koja je poslana od strane odašiljača s već prije definiranim parametrima objekte predane kao argument toj metodi. Ukoliko da, metoda će vratiti ne prazno polje podataka u kojima je navedeno koji su objekti presječeni zrakom te u kojoj točci je sjecište. Objekti su poredani od onog najbližeg izvoru zrake do onog najdaljeg. Tako pomoću odašiljača zraka znamo je li u trenutku pritiska tipke miša pozicija pokazivača bila nad željenim dijelom modela ili nije.

4.4.4. Detector.js

Cilj skripte Detector.js [37] je provjeriti podržava li korišteni internetski preglednik WebGL, te ukoliko ne podržava, ispisuje poruku o tome što učiniti kako bi se u internetskom pregledniku to omogućilo.

4.4.5. Stats.js

Stats.js [38] skripta očitava brzinu osvježavanja prikaza aplikacije koja koristi Three.js te prikazuje podatke u odnosu na brzinu. Može prikazati broj okvira po sekundi ili broj milisekundi potreban za osvježavanje prikaza jednog okvira. ***

4.4.6. Tween.js

Naziv Tween.js [39] biblioteke dolazi od engleskih riječi „in-between“ što bi se moglo prevesti kao „u međuvremenu“. Taj opis odlično odgovara ovoj biblioteci i njezinoj zadaći. Cilj ove biblioteke je za zadanu početnu i konačnu vrijednost neke varijable (ili samo konačnu) odrediti one vrijednosti varijable koje će se pojavljivati slijedno između te dvije vrijednosti, ako proces promjene iz početne vrijednosti u konačnu mora trajati određeno vrijeme. Tween.js biblioteka se tako najčešće koristi za glatke 2D i 3D animacije u web aplikacijama jer je dosta pokreta u grafici često definirano samo početnom i konačnom točkom. Ukoliko je pokret kompliciraniji, često se može ulančati kao niz manjih pokreta koji imaju svoju početnu i konačnu točku te koji se međusobno nadovezuju. Ulančavanje nudi i biblioteka Tween.js. U slučaju korištenja Tween.js zajedno s Three.js bibliotekom, kako je slučaj u aplikaciji MSMA, kreirat ćemo novi *Tween* objekt te mu kao varijablu promjene predati atribut objekta koji želimo promijeniti. Programski kod 7. prikazuje takav primjer stvaranja objekta gdje je željena transformacija rotacija te se kao varijabla promjene predaje vrijednost Eulerovih kutova gornjeg poklopca Micro Soot modela. Zatim se definira konačna točka i period koji će proći dok vrijednost varijable ne dođe do konačne. U ovom slučaju je naveden i period odgode koji će proći prije nego što se vrijednost varijable krene mijenjati. Isječak će nakon pola sekunde početi mijenjati kut rotacije objekta od početne vrijednosti do kuta -90° u odnosu na x os kroz period od jedne sekunde. Kako bi se promjene zaista i događale, moramo označiti početak promjene vrijednosti varijable pozivom metode *start* stvorenog objekta, te prilikom osvježavanja prikaza pozivati *update* prostora imena (engl. namespace) *TWEEN.*

Tim pozivom svi pokrenuti *Tween* objekti ažuriraju vrijednosti varijabli koje su im predane.

U MSMA aplikaciji Tween.js biblioteka se koristila u tri načina rada: „Explode“, „Teach Me“ i „Test Me“. U načinu rada „Explode“ biblioteka se koristila na način da je za svaku eksploziju bio definiran konačan položaj do kojeg je dio Micro soot modela morao doći, bez obzira na to na kojem je položaju bio prije toga. U načinima rada „Teach Me“ i „Test Me“ Tween.js se koristi kako bi se prikazao odgovarajući korak u procesu održavanja uređaja Micro Soot.

```
var rotation animation = new TWEEN.Tween(
                objDict["ms_LidUpper"].rotation
        )
          .to( {x:-1.571},1000)
          .delay(500);
```

Programski kod 7. Primjer kreiranja Tween elementa

4.4.7. TrackballControls.js

Biblioteka TrackballControls.js [40] je biblioteka koja omogućuje funkcionalnosti dostupne u načinu rada „Free Camera“: rotacija kamere (engl. rotate), translacija kamere zajedno s translacijom točke pogleda (engl. pan) i približavanje/udaljavanje kamere od njezine točke pogleda (engl. zoom). Biblioteka sadrži samo jedan razred *THREE.TrackballControls* (u Javascript jeziku tretiranu kao funkciju) koja kao konstruktor prima *THREE.Camera* objekt koji je kamera scene, te dom element u koji se scena iscrtava. Programski kod 8. pokazuje primjer stvaranja jednog takvog razreda.

Biblioteka radi tako da čeka na događaje (engl. event) miša te ovisno o tipu događaja prelazi u novo stanje. Tako se pritiskom lijeve tipke miša prelazi u stanje „Rotate“, pritiskom desne tipke miša u stanje „Pan“, a pomicanjem „kotačića“ miša (što se tretira kao pritisak srednje tipke) u stanje „Zoom“. Brzina bilo koje od spomenute tri akcije ovisi o kretanju miša nakon što se ostvario jedan od tri događaja. Razred interno čuva prijašnju vrijednost pokazivača te je uspoređuje s trenutnom vrijednosti. Ako su razlike između prijašnje i trenutne vrijednosti pokazivača veće, akcija će imati jači utjecaj, i obrnuto. Smjer kretanja miša isto tako utječe na to kako se akcija obavlja. Tako se na primjer u stanju „Rotate“, pomakom miša u lijevu stranu, kamera vrti u desno, a pomakom miša prema gore, kamera vrti prema dolje. Unutar primjera Programski kod 8. može se vidjeti da je dopuštena

regulacija brzine obavljanja svake od akcija. Kako bi u aplikaciji poput MSMA korištenje *THREE.TrackballControls* razreda imalo utjecaja, potrebno je periodički pozivati metodu *update* nad instancom tog razreda kako bi se položaj pokazivača često ažurirao. Najbolje mjesto za taj poziv je unutar *render* funkcije baš kao što se ažurirala vrijednost rotacije u primjeru Programski kod 6.

```
//camera
camera = new THREE.PerspectiveCamera(fov, width/height, 0.1, 500);
camera.position = new THREE.Vector3(0,-10,60);

//renderer
renderer = new THREE.WebGLRenderer( {antialias: true} );
renderer.setSize(width, height);

//controls
controls = new THREE.TrackballControls( camera, renderer.domElement );
controls.rotateSpeed = 1.0;
controls.zoomSpeed = 1.2;
controls.panSpeed = 0.8;
```

Programski kod 8. Stvaranje instance *TrackballControls* razreda

5. Ugrađivanje podrške upravljanja uređajem Leap Motion u MSMA aplikaciju

U ovom poglavlju bit će prikazan način na koji se ostvarilo upravljanje MSMA aplikacije pomoću uređaja Leap Motion. Cilj je bio ostvariti sve funkcionalnosti koje pruža MSMA aplikacija tako da se mogu ostvariti pokretima ruke koji će se prepoznati preko uređaja Leap Motion.

5.1. Dodavanje modela ruku unutar Three.js aplikacije

Kako bi se dodao model ruku unutar Three.js aplikacije, korišten je dodatak *riggedHand* (poglavlje 3.4. ovoga rada). Atribut opcija *positionScale* daje mogućnost skaliranja položaja ruke na način da se položaj množi s brojem koji se predaje kao vrijednost. Pošto se unutar MSMA aplikacije položaj kamere često mijenja, bilo je potrebno postaviti kameru kao objekt roditelj rukama koje se prikazuju. Kako bi se to postiglo, korišten je dodatak *transform* (poglavlje 3.4. ovoga rada) kojemu su kao opcije predana pozicija ruku u odnosu na objekt roditelj te skala s kojom se množi veličina ruku kako bi veličina ruku bila prikladna sceni u kojoj se nalazi. Programski kod 9. prikazuje dodavanje spomenutih dodataka. Atributi *scale* i *position* dodatka *transform* te atribut *positionScale* dodatka *riggedHand* su regulirani ovisno o udaljenosti kamere od točke pogleda kako položaj ili brzina ruke ne bi bili nesukladni poziciji u sceni na kojoj se kamera nalazi.

```
controllerLeap = new Leap.Controller({
    enableGestures: true
})
    //dodatak transform
    .use('transform', {
        quaternion: new THREE.Quaternion(),
        position: new THREE.Vector3(0, -5, -20),
        effectiveParent: camera,
        scale: 0.03
    })
    //dodatak riggedHand
    .use('riggedHand', {
        parent: scene,
        scene: scene,
        camera: camera,
        renderer: renderer,
        renderFn: function () {
            renderer.render(scene, camera);
        },
        positionScale: 2
    });
```

Programski kod 9. Stvaranje instance *TrackballControls* razreda

5.2. Prilagođavanje TrackballControlls.js biblioteke za LeapMotion

TrackballControlls.js biblioteka (poglavlje 4.4.7. ovoga rada) koja nudi mogućnost rotiranja kamere, pomicanja kamere i njene točke pogleda te promjene udaljenosti između kamere i njezine točke pogleda korištena je u prva tri od pet načina rada MSMA aplikacije. Rad biblioteke ovisi o trenutnom stanju u kojem se nalazi te o pomicanju računalnog miša. Kako bi se biblioteka prilagodila uređaju Leap Motion, onemogućena su sva ažuriranja internih varijabli za trenutnu i prijašnju vrijednost položaja pokazivača svaki puta kada se unutar vidnog polja uređaja Leap Motion nalazi barem jedna ruka. Umjesto toga, varijable se ažuriraju na temelju položaja ruku unutar vidnog polja.

Tijekom izrade funkcionalnosti rotacije kamere korištenjem Leap Motion-a, naišlo se na problem da se rotirati u jednu od strana moglo samo do 90°, a kasnije bi rotacija bila ili otežana, ili bi se ovisno o rotaciji moralo pomicati ruku u suprotnom smjeru kako bi se kamera nastavila dalje rotirati. Krivac tome je dodatak *transform* koji transformira položaj ruke na način da umjesto vrijednosti koje predstavljaju udaljenost ruke od uređaja Leap Motion, postavi položaj ruke unutar Three.js scene. Problem je bio taj što se prilikom rotiranja kamere, ruke rotiraju zajedno s kamerom, pošto je kamera njihov objekt roditelj, te se nakon rotacije od 90° stupnjeva, s korisnikovog gledišta, os *x* i os *z* zamijene. Tako da nakon te rotacije pomak ruke u desno u stvarnom svijetu ne mijenja os *x* unutar Three.js scene, kako bi to možda bilo očekivano, nego os *z*, dok vrijednosti na *x* osi ostaju gotovo nepromijenjene. Kako bi se to riješilo, dodatak *transform* je izmijenjen na način da zapamti vrijednost položaja ruke u stvarnom svijetu prije nego napravi transformaciju položaja, tako da su oba, prvotni i transformirani položaj ruke, dostupni u svakom okviru.

Za ažuriranje trenutnih i prijašnjih vrijednosti položaja ruke koje prati razred *THREE.TrackballControls* nisu se koristile točne vrijednosti položaja ruke, nego normalizirane vrijednosti. Za normalizaciju, korišten je razred IntractionBox (poglavlje 3.2.5. ovoga rada).

Funkcionalnost promjene položaja kamere zajedno s položajem točke pogleda nije ostvarena zbog toga što bi ispravan rad te funkcionalnosti prilikom korištenja Leap

Motion-a zahtjeva dublje poznavanje o tome kako radi dodatak *transform* te TrackballControls.js biblioteka.

5.3. Pomicanje dijelova modela Micro Soot u načinu rada „Disassemble“

Kako bi se ostvarilo pomicanje dijelova modela Micro Soot u načinu rada „Disassemble“ koristio se razred *Raycaster* biblioteke Three.js (4.4.3. poglavlje ovoga rada). Kao parametar za poziciju iz koje je zraka odaslana je postavljen položaj dlana ruke, a kao smjer odašiljanja zrake postavljena je normala dlana. Nakon što je zraka odaslana, provjerava se je li prošla kroz ijedan od dijelova Micro Soot modela, te ako je, najbliži od njih, gledajući od ruke, je uhvaćen. Hvatanje objekta okida se gestom ruke (pogledati 5.5.2. poglavlje ovoga rada) čime objekt počinje biti vezan za ruku te se pomiče u sceni prateći njene pokrete. Objekt se prilikom hvatanja ne pomiče točno na poziciju ruke, nego je u trenutku hvatanja izračunat razmak između ruke i objekta koji je tijekom pomicanja ruke uvijek konstantan. Izvršavanjem iste geste nanovo ili pomicanjem ruke u stranu toliko da izađe izvan vidnog polja uređaja Leap Motion će otpustiti objekt na poziciji na kojoj se trenutno nalazi. Time objekt prestaje pratiti pokrete ruke te se za njegovo ponovno hvatanje postupak mora ponoviti.

5.4. Način rada „Test Me“

Kako bi se prošao cijeli postupak održavanja u načinu rada „Test Me“ korištenjem uređaja Leap Motion, koristio se razred *Raycaster* biblioteke Three.js kao i u načinu rada „Disassemble“. Razred se inicijalizirao na isti način kako je to napravljeno prilikom hvatanja objekta u načinu rada „Disassemble“, uz to da su mu uz ishodišta zrake i smjera zrake dan i argument koji određuje koliko daleko zraka ide. Time se dobilo da ruka mora biti blizu objekta koji sudjeluje u trenutnom koraku postupka održavanja uređaja Micro Soot, nasuprot načinu rada „Disassemble“ gdje udaljenost ruke i objekta koji se hvata nije važna. U ovom načinu rada, nije potrebno učiniti gestu da bi se stvorio odašiljač zraka i poslala sama zraka, nego se on stvara u svakom okviru nanovo te se zraka odašilje u smjeru normale dlana ruke. Za razliku od načina rada „Disassemble“, u provjeravanju je li zraka presjekla objekt, ne

provjeravaju se svi dijelovi modela Micro Soot, nego samo oni koji sudjeluju u trenutnom koraku postupka tako da se performanse aplikacije nisu pogoršale.

Kako bi hvatanje objekta unutar načina rada „Disassemble“ te doticanje objekta unutar načina rada „Test Me“ bilo olakšano, modelima ruku su tijekom rada aplikacije nacrtane normale dlanova na koordinatama položaja dlana kako bi simulirale zraku koju odašiljač zraka šalje (Slika 20.).

a)

b)

Slika 20. a) model ruke bez normale, b) model ruke s nacrtanom normalom

Za razliku od korištenja aplikacije s računalnim mišem gdje se animacija postupka okidala pritiskom lijeve tipke miša, u ovom slučaju se okida samo u ovisnosti o položaju ruke. To znači da se u slučaju da je isti dio modela zaslužan za dva ili više uzastopna koraka postupka održavanja uređaja, animacije okidaju jedna za drugom bez izvršavanja do kraja. Razmak između dvije animacije bio bi jedan okvir te bi se u nizu koraka, gdje je isti model zaslužan za okidanje animacije, izvršila samo posljednja animacija. Kako bi se to izbjeglo, okidanje novog koraka postupka unutar „Test Me“ načina rada obavlja se samo u slučaju određene vrijednosti zastavice. Vrijednost zastavice se prilikom okidanja novog koraka postupka postavlja na 0 te se korištenjem Tween.js biblioteke (4.4.6. poglavlje ovoga rada) kroz određeni period vraća na početnu vrijednost. Promjena vrijednosti zastavice se izvršava u isto vrijeme kada i animacija postupka, tako da se samo animacije koje traju puno duže u usporedbi s ostalima mogu prekinuti okidanjem novog koraka postupka.

5.5. Razred GestureRecognizer.js

Kako bi se uređajem Leap Motion upravljalo MSMA aplikacijom potrebno je tijekom svakog okvira koji se obrađuje, provjeriti je li korisnik napravio neku od gesti te učiniti odgovarajuću akciju ovisno o dobivenom odgovoru te o tipu geste koji se dogodio. Unutar leap.js biblioteke prepoznaju se četiri različite geste, no kako bi se korisniku dalo više mogućnosti, neke od gesti su specificirane na način da su osim njihovog tipa bitni i neki od ostalih atributa te će se tako u istom stanju aplikacije, za isti tip geste, ovisno o vrijednostima atributa, okinuti druga akcija.

5.5.1. Gesta Swipe

Unutar MSMA aplikacije, prepoznaje se 4 različitih vrsta gesti *Swipe,* pri tome tri horizontalne i jedna vertikalna: pokret lijeve ruke prema desno te pokreti desne ruke prema lijevo, desno i dolje. Prva bitna komponenta u određivanju vrste ove geste je smjer pokreta ruke. Određivanje smjera se radi pomoću vektora smjera *direction* gdje se uspoređuje apsolutna vrijednost širine s apsolutnom vrijednošću visine te se na temelju te usporedbe može znati je li pokret horizontalan ili vertikalan. Zatim se u ovisnosti o predznaku, veće od spomenute dvije vrijednosti, donese zaključak o točnom smjeru geste. Programski kod 10. prikazuje određivanje smjera geste *Swipe*. Druga važna komponenta u određivanju vrste geste je ruka kojom se gesta radi.

```
var getSwipeDirection = function (gesture) {
    //provjera je li smjer horizontalni
    //tako da gleda x i y koordinatu jediničnog vektora smjera geste
    var isHorizontal = Math.abs(gesture.direction[0]) >
        Math.abs(gesture.direction[1]);

    //određivanje točnog smjera (lijevo, desno, gore ili dolje)
    if (isHorizontal) {
        if (gesture.direction[0] > 0) {
            return _this.SWIPE_DIRECTION.RIGHT;
        } else {
            return _this.SWIPE_DIRECTION.LEFT;
        }
    } else { //vertical
        if (gesture.direction[1] > 0) {
            return _this.SWIPE_DIRECTION.UP;
        } else {
            return _this.SWIPE_DIRECTION.DOWN;
        }
    }
};
```

Programski kod 10. Određivanje smjera *Swipe* geste [41]

Dvije činjenice su važne prilikom obrađivanja geste *Swipe*. Prva je da svaki prst izvršava gestu *Swipe* nezavisno od ostalih prstiju ruke, a druga je da se svaka gesta *Swipe* izvršava kroz nekoliko različitih okvira zato što mora proći kroz tri stanja, s time da završava u stanju „stop“ (poglavlje 3.3.4. ovoga rada). Iz toga proizlazi da prilikom izvršavanja geste *Swipe* s otvorenom rukom, što znači da svih pet prstiju zasebno izvršava svoju gestu *Swipe*, nije nužno da će sve geste završiti u istom okviru. To znači da ako želimo pratiti koliko prstiju ruke je napravilo gestu Swipe, mora se promatrati period period od nekoliko okvira, a ne samo jedan okvir zasebno. Eksperimentiranjem unutar MSMA aplikacije se zaključilo da je broj okvira potrebnih, kako bi svih pet gesti *Swipe* (svaka za jedan prst) završilo, uvijek 4 nakon završetka prve od gesti, stoga se broj 5 uzeo kao raspon okvira koji se promatra prilikom obrade ove geste.

Horizontalna gesta *Swipe* se definirala tako da su unutar 5 okvira tri različita prsta generirala gestu u istom smjeru (lijevo ili desno), te da je pokret obavljen istom rukom. Pokret desno i lijevo desnom rukom služili su za promjenu načina rada unutar MSMA aplikacije, a pokret desno lijevom rukom za pokretanje načina rada „Teach Me“ i „Test Me“. Ponovljenim pokretom lijevom rukom u desno se izlazi iz spomenutih načina rada.

Vertikalna gesta *Swipe,* tj. pokret desne ruke prema dolje, se koristila u „Explode“ načinu rada MSMA aplikacije. Promatralo se koliko je prstiju generiralo gestu *Swipe*

desnom rukom prema dolje. Ovisno o broju prstiju koji su generirali gestu *Swipe*, okinula se različita eksplozija, gdje je za jedan prst okinut proces vraćanja modela u početno stanje, a za broj prstiju veći od jedan pripadajuća eksplozija. Pošto različitih eksplozija ima 4, broj prstiju jedne ruke je dovoljan da pokrije sve moguće slučajeve.

5.5.2. Gesta KeyTap

Unutar MSMA aplikacije, prepoznaje se dvije vrste geste *KeyTap*. Razlika između gesti je prst koji obavlja gestu, a gestu se prepoznavalo ako ju je obavio kažiprst ili srednji prst. Programski kod 11. pokazuje način na koji se iz geste određuje prst koji obavlja gestu. U kodu je pretpostavljeno da će objekt koji izvršava gestu uvijek biti prst, kako je zamišljeno unutar MSMA aplikacije, a ne neki drugi objekt. Ova gesta se pojavljuje samo u jednom stanju, stanju „stop“ te je onda dovoljno promatrati samo trenutni okvir kako bi se gesta pronašla.

```
//odredi identifikacijsku oznaku objekta koji je izvršio gestu
var pointableId = gesture.pointableIds[0];
var pointable = frame.pointable(pointableId);

//pretvori objekt u tip Finger
var finger = new Leap.Finger(pointable);

if (finger.type == _this.FINGER_TYPES.INDEX) {
    //ako je kažiprst učini...
} else if (finger.type == _this.FINGER_TYPES.MIDDLE) {
    //ako je srednji prst učini...
}
```

Programski kod 11. Određivanje tipa prsta

Gesta *KeyTap* obavljena kažiprstom u načinu rada „Disassemble“ obavlja hvatanje dijela Micro Soot modela. Unutar načina rada „Teach Me“ ista gesta će okinuti animaciju koraka postupka koji prethodi trenutnom.

Gesta KeyTap obavljena srednjim prstom će unutar načina rada „Teach Me“ okinuti animaciju koraka postupka koji slijedi poslije trenutnog.

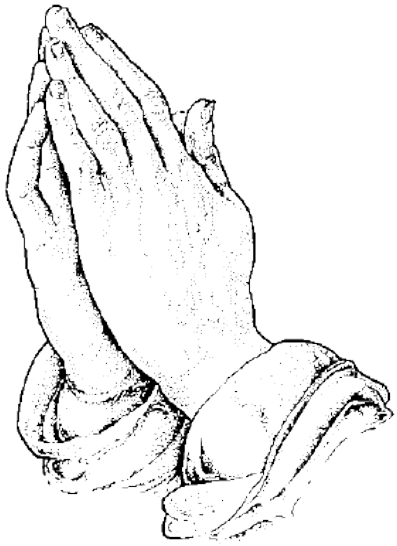
Gesta *Swipe* u smjeru prema dolje vrlo često će okinuti gestu *KeyTap* zbog sličnosti načina na koji gesta *Swipe* završava i same *KeyTap* geste. Zbog toga se gesta *KeyTap* neće nikada očitati ako je unutar trenutnog okvira prepoznata gesta *Swipe* u bilo kojem stanju kako bi se izbjegnule kolizije.

5.5.3. Gesta ScreenTap

Kao i gesta *KeyTap,* gesta *ScreenTap* se pojavljuje samo u jednom stanju, stanju „stop“ te je dovoljno promatrati samo trenutni okvir kako bi se gesta pronašla. Postoji samo jedna vrsta ove geste unutar MSMA aplikacije, a to je gesta *ScreenTap* obavljena kažiprstom. Gesta se koristi unutar načina rada „Teach Me“ tako da okida animaciju trenutnog koraka postupka, a unutar načina rada „Test Me“ tako da korisniku ispisuje poruku pomoći za trenutni korak postupka.

5.5.4. Gesta ClaspedHands

Gesta *ClaspedHands* (hrv. sklopljene ruke) je jedina gesta u kojoj sudjeluju obje ruke istovremeno. Gesta je izvršena tako da su dlanovi ruku usmjereni jedan prema drugome na udaljenosti ne većoj od 5 centimetara (Slika 21.). Gesta je implementirana na način da su se zbrojile normale dlanova. Zbroj se usporedio s malim brojem te ukoliko je zbroj bio manji, provjerila se udaljenost. U slučaju zadovoljenja uvjeta udaljenosti, gesta je bila ostvarena. Gesta se koristi za resetiranje aplikacije u početno stanje.



Slika 21. Gesta *ClaspedHands* [42]

5.5.5. Geste Grab i Pinch

Za geste *Grab* i *Pinch* iskoristili su se atributi razreda *Hand* (poglavlje 3.2.3. ovoga rada) *grabStrenght* i *pinchStrenght*. Uređaj Leap Motion jako dobro prepoznaje kada je ruka u položaju *grab*, stoga će gesta *Grab* biti ostvarena kada

je *grabStrenght* veći od 0.99. Položaj ruke *pinch* koji se definira kao snaga dodira između palca i kažiprsta je često u koliziji s položajem *grab*. Kako bi se kolizija izbjegla, gesta *Pinch* se definira tako da atribut *pinchStrenght* mora biti veći od 0.95, a istovremeno atribut *grabStrenght* mora biti manji od 0.4. U načinima rada gdje su navedene funkcionalnosti omogućene, gesta *Grab* se koristi za rotaciju kamere oko njezine točke pogleda, a gesta *Pinch* za približavanje ili udaljavanje kamere od njezine točke. Obje geste se koriste na način da je aplikacija u stanju funkcionalnosti koju okidaju sve dok se gesta očitava. Kada se gesta na ruci prestane očitavati ili ruka izađe izvan vidnog polja uređaja Leap Motion, izlazi se iz stanja funkcionalnosti u „normalno“ stanje.

6. Zaključak

Uređaj Leap Motion je uređaj kojim se promatraju pokreti korisnikovih ruku te je korak naprijed u interakciji čovjeka i računala. Uređaj je malen i lako prenosiv te ima impresivne specifikacije. Unatoč tomu, kako je u poglavlju 2.4. rečeno, u trenucima kada su ruke na rubovima vidnog polja, performanse mu znatno variraju. Slabiji rezultati na rubovima vidnog polja su se primijetili i tijekom izrade praktičnog dijela ovoga rada, kada je zadatak uređaja bio prepoznavanje gesti, no u velikoj većini slučajeva uređaj radi vrlo dobro. Duže korištenje uređaja može biti zamorno zbog neprirodnog položaja ruku, stoga je potrebno držati se uputa korištenja (poglavlje 2.3.) kako bi se umor umanjio.

U ovom radu cilj je bio iskoristiti Leap Motion aplikacijsko programsko sučelje kako bi se pomoću gesti prepoznatih uređajem upravljalo 3D objektom. Promatrani objekt je bio 3D model Micro Soot, koji predstavlja uređaj za kontinuirano mjerenje koncentracije čađi u ispušnim plinovima dobivenih iz motora s unutarnjim izgaranjem. Njime se upravljalo kroz web aplikaciju Micro Soot maintenance app tako da su se u spomenutu aplikaciju ugradile funkcionalnosti Leap Motion-a. Aplikacijsko programsko sučelje Leap Motion je hijerarhijski strukturirano s naglaskom na razred *Frame* koji je izvor svih podataka tijekom osvježavanja prikaza. Leap.js biblioteka omogućuje jednostavan način integracije Leap Motion funkcionalnosti unutar web aplikacije koja koristi Three.js biblioteku. Uz dodatke leap.js biblioteci, jednostavno je dodati vidljivi model ruku u scenu te vezati taj model za druge objekte u sceni.

Iako je skup gesti koje sam uređaj prati dosta ograničen, moguće je povećati taj broj na dva načina. Prvi je način analiziranje parametara gesti koje uređaj Leap Motion već prepoznaje te na temelju njihovih vrijednosti odrediti akciju. Drugi način je iskorištavanje podataka dobivenih iz okvira kako bi se definirale vlastite geste. Da bi se prepoznalo okidanje gesti na temelju njihovih uvjeta okidanja potrebno je implementirati vlastiti prepoznavatelj gesti. Prilikom upravljanja aplikacijom implementiranim gestama, treba se paziti na kolizije između gesti kako bi se olakšalo korištenje aplikacije te smanjio broj slučajnih okidanja neželjenih akcija.

_____

Damir Ciganović-Janković

7. Literatura

[1] Mary Bells, The History of Eniac Computer, http://inventors.about.com/od/estartinventions/a/Eniac.htm, pristupljeno 14.06.2015.

[2] Wikipedia, Power Glove, https://en.wikipedia.org/wiki/Power_Glove, pristupljeno 14.06.2015.

[3] Intel Corporation, Intel RealSense Overview, http://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html, pristupljeno 14.06.2015.

[4] Microsoft, Kinect for Windows, https://www.microsoft.com/en-us/kinectforwindows/, pristupljeno 14.06.2015.

[5] Leap Motion, Inc, Leap Motion Controller, https://www.leapmotion.com/product, pristupljeno 14.06.2015.

[6] Wikipedia, XBOX https://en.wikipedia.org/wiki/Xbox, pristupljeno 14.06.2015.

[7] Leap Motion, Inc, Leap Motion Company, https://www.leapmotion.com/company, pristupljeno 15.06.2015.b

[8] Oculus VR, LLC, Oculus https://www.oculus.com/en-us/, pristupljeno 15.06.2015.

[9] Alex Colgan, How Does the Leap Motion Controller Work?, nastanak 09.08.2014., http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/, pristupljeno 15.06.2015.

[10] Wikipedia, Infrared, https://en.wikipedia.org/wiki/Infrared, pristupljeno 15.06.2015.

[11] Kyle Hay, Designing the Leap Motion Controller, nastanak 03.04.2013., http://blog.leapmotion.com/designing-leap-motion-controller/, pristupljeno 15.06.2015.

[12] Leap Motion, Inc, Leap Motion Controller Important Information Guide, http://lm-assets.s3.amazonaws.com/legal/Important-Information-Guide-130411.pdf, pristupljeno 15.06.2015.

[13] RobotShop inc, Leap Motion 3D Motion Controller, http://www.robotshop.com/en/leap-motion-3d-motion-controller.html, pristupljeno 15.06.2015.

[14] Jože Guna, Grega Jakus, Matevž Pogačnik, Sašo Tomažić, Jaka Sodnik, An Analysis of the Precision and Reliability of the Leap Motion Sensor and Its Suitability for Static and Dynamic Tracking, nastanak 21.02.2014., https://www.mdpi.com/1424-8220/14/2/3702/pdf, pristupljeno 16.06.2015.

[15] Leap Motion, Inc, System Architecture, https://developer.leapmotion.com/documentation/javascript/devguide/Leap_Architecture.html, pristupljeno 16.06.2015.

[16] Leap Motion, Inc, LeapJS, https://github.com/leapmotion/leapjs, pristupljeno 16.06.2015.

[17] Leap Motion, Inc, API Overview, https://developer.leapmotion.com/documentation/javascript/devguide/Leap_Overview.html, pristupljeno 16.06.2015.

[18] Leap Motion, Inc, Tracking Model, https://developer.leapmotion.com/documentation/javascript/devguide/Leap_Tracking.html, pristupljeno 17.06.2015.b

[19] Peter Erlich, pinchStrenght, nastanak 15.04.2014., https://developer.leapmotion.com/gallery/pinchstrength, pristupljeno 17.06.2015.

[20] Raimo Tuisku, Grab Strenght, nastanak 15.04.2014., https://developer.leapmotion.com/gallery/grab-strength, pristupljeno 17.06.2015.

[21] Leap Motion, Inc, Coordinate Systems, https://developer.leapmotion.com/documentation/csharp/devguide/Leap_Coordinate_Mapping.html, pristupljeno 18.06.2015.

[22] Leap Motion, Inc, Gestures, https://developer.leapmotion.com/documentation/javascript/devguide/Leap_Gestures.html, pristupljeno 19.06.2015.

[23] Peter Erlich, LeapJS Plugins, nastanak 01.04.2014., https://github.com/leapmotion/leapjs-plugins, pristupljeno 19.06.2015.

[24] Peter Erlich, JS Rigged Hand Plugin, nastanak 02.02.2014., https://github.com/leapmotion/leapjs-rigged-hand, pristupljeno 19.06.2015.

[25] Ana Nekić, Grafički objekti u Web preglednicima, nastanak u lipnju 2013., https://bib.irb.hr/datoteka/636619.Grafiki_objekti_u_Web_preglednicima.pdf, pristupljeno 19.06.2015.

[26] Marko Pilipović, Tehnike interakcije 3D objektima u web preglednicima, nastanak u lipnju 2013., http://www.zemris.fer.hr/predmeti/ra/Magisterij/13_Pilipovic/diplomski.pdf, pristupljeno 20.06.2015.

[27] AVL, MSSplus – AVL Micro Soot Sensor, https://www.avl.com/-/mssplus-avl-micro-soot-sensor, pristupljeno 20.06.2015.

[28] AVL, Operating Manual Product Guide, nastanak u svibnju 2008., http://www.me.umn.edu/centers/cdr/reports/AVLmanual2008.pdf, pristupljeno 20.06.2015.

[29] Mozilla Developer Network and individual contributors, HTML (Hyper Text Markup Language), https://developer.mozilla.org/en-US/docs/Web/HTML, pristupljeno 21.06.2015.

[30] Javascript.com, Not exactly sure what javascript is?, https://www.javascript.com/, pristupljeno 21.06.2015.

[31] Duncan Aitken, What is HTML5?, http://www.html-5-tutorial.com/about-html5.htm, pristupljeno 21.06.2015.

[32] Mozilla Developer Network and individual contributors, WebGL, https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API, pristupljeno 21.06.2015.

[33] Khronos Group, OpenGL ES, https://www.khronos.org/opengles/, pristupljeno 21.06.2015.

[34] Alexis Deveria, Can I use WebGL – 3D Canvas graphics, http://caniuse.com/#feat=webgl, pristupljeno 22.06.2015.

[35] Marko Gaćeša, WebGL + THREE.js 3D grafika na mreži, nastanak 26.05.2014., http://www.slideshare.net/PSTechSerbia/webgl-andthreejs, pristupljeno 22.06.2015.

[36] Introducing JSON, http://json.org/, pristupljeno 22.06.2015.

[37] Doug Fritz, Detector.js, nastanak 05.05.2011., https://code.google.com/r/fxnstyling-webgl-globe-autospin/source/browse/globe/Three/Detector.js, pristupljeno 23.06.2015.

[38] Ricardo Cabello, stats.js, nastanak 04.04.2010., https://github.com/mrdoob/stats.js/, pristupljeno 23.06.2015.

[39] Soledad Penades, Ricardo Cabello, nastanak 23.05.2010., https://github.com/tweenjs/tween.js/, pristupljeno 23.06.2015.

[40] Ricardo Cabello, TrackballControls.js, nastanak 04.10.2012., https://github.com/mrdoob/three.js/blob/master/examples/js/controls/TrackballControls.js, pristupljeno 24.06.2015.

[41] Leap Motion, Inc, SwipeGesture, https://developer.leapmotion.com/documentation/javascript/api/Leap.SwipeGesture.html, pristupljeno 24.06.2015.

[42] Letters From a Young Catholic, nastanak 28.05.2007., http://dilexitprior.blogspot.com/2007/05/praying-for-those-who-do-not.html, pristupljeno 24.06.2015.

Upravljanje gestama 3D objektom pomoću uređaja Leap Motion

8. Sažetak

Leap Motion je uređaj za prepoznavanje pokreta i gesti ruku. U radu je objašnjen način na koji uređaj radi te koje su njegove prednosti i nedostaci. Glavni cilj rada bio je iskoristiti aplikacijsko programsko sučelje Leap Motion-a, biblioteku leap.js, kako bi se ostvarilo upravljanje 3D objektom unutar aplikacije. Cilj je ostvaren ugrađivanjem podrške za Leap Motion funkcionalnosti unutar već postojeće web aplikacije Micro Soot maintenance app tako da se aplikacijom upravljalo pomoću gesti. Aplikacija ima pet različitih načina rada te je za njenu implementaciju korišteno nekoliko različitih Javascript biblioteka, od kojih je posebno važna Three.js potrebna za implementaciju 3D grafike unutar aplikacije. Za potrebe prepoznavanja gesti bilo je potrebno implementirati vlastiti prepoznavatelj gesti kako bi se proširio ograničen skup gesti koje sam uređaj može prepoznati. Gestama se aplikacijom upravljalo na način da je svaka gesta, ovisno o načinu rada, okidala neku od akcija unutar aplikacije, ponekad zavisno o položaju ruke, a ponekad ne.

Ključne riječi: Leap Motion, leap.js, prepoznavanje pokreta ruke, prepoznavanje gesti ruke, Three.js

Gesture based interaction using Leap Motion device

9. Abstract

Leap Motion is a device for hand motion and hand gesture recognition. This paper explains how Leap Motion works and what are its advantages and disadvantages. The main objective of this study was to utilize Leap Motion application programming interface (leap.js library) in order to manage 3D object within an application by using Leap Motion device. The goal is achieved by incorporating support for Leap Motion functionality within existing web application, Micro Soot maintenance app, to manage the application by using gestures. The application has five different modes. Number of different Javascript libraries are used for its implementation, with Three.js library as particularly important because it was needed to implement 3D graphics within the application. For the purposes of gesture recognizing, gesture recognizer was needed to be implemented to spread a limited set of gestures that the device can recognize. Application was managed by gestures in such a way that every gesture, depending on the application mode, was triggering some of the actions within the application, sometimes depending on the hand position, sometimes not.

Keywords: Leap Motion, leap.js, hand motion recognition, hand gesture recognition, Three.js