Moderne tehnike interakcije s 3D objektima

Tehnička dokumentacija

Verzija 1.0

Studentski tim: Hrvoje Gazibara

Matejka Ivančić

Damjan Križaić

Janko Sladović

Andrija Stepić

Marijan Šuflaj

Marko Vrljičak

Nastavnik: Željka Mihajlović

Sadržaj

1. Uvod

S napretkom tehnologije aplikacije više ne primaju ulazne informacije samo preko miša i tipkovnice za jednostavne aplikacije za stolna računala, već je potrebno razvijati prijenosne aplikacije za različite platforme, različite tipove uređaja s različitim ulazima.

Najpoznatiji ulazni uređaji su i dalje tipkovnica i miš, no s današnjim modernim prijenosnim uređajima krenulo se u vode znanstvene fantastike, pa tako korisnici mogu „komunicirati“ s uređajima preko dodirnih gesti, pokretima ruku (3D gesti) i govornih naredbi.

Problem je razviti aplikacije koje se mogu koristiti na raznovrsnim/različitim platformama i implementirati različite metode obrade ulaza nastalih interakcijom na precizan i uniformni način kao što se može i s klasičnim ulaznim uređajima (tipkovnicom i mišem).

Cilj ovog projekta je proučiti koje tehnologije najbolje podržavaju razvoj grafičkih aplikacija za različite platforme (Mac OS X, Microsoft Windows, Linux i dr.), te proučiti moderne tehnologije ulaznih uređaja kao što su dodirne geste, 3D geste pomoću uređaja Leap Motion i govorne naredbe, te razviti aplikaciju koja će implementirati navedene metode interakcije s aplikacijama i vidjeti koliko su te metode pouzdane za kontroliranje 3D grafičkih aplikacija.

Rezultat projekta je program koji predstavlja prototip interaktivnog 3D grafičkog priručnika u industriji, a omogućuje interakciju pomoću navedenih različitih ulaza.

2. Korištene tehnologije

U sklopu projekta proučena je kontrola interakcije s aplikacijom pomoću gesti ruku i uređaja za detekciju Leap Motion, glasovnih naredbi te dodirnih gesti u *Unity Engine* i web okruženju.

Unity je set alata za izradu računalnih igara i drugog interaktivnog sadržaja. Nudi integrirano grafičko razvojno okruženje koje skriva kompleksnost pokretačkog mehanizma čije se funkcionalnosti koriste za izradu igre , a ipak daje kontrolu nad istim putem intuitivnih alata.

Najveći argumenti za korištenje u projektu bila je mogućnost široke primjene preko različitih. Igre je moguće razvijati na Windows i MAC platformama za Windows, MAC, Linux, Wii, iPad, iPhone, Android, XBOX 360 i PlayStation 3 platforme.

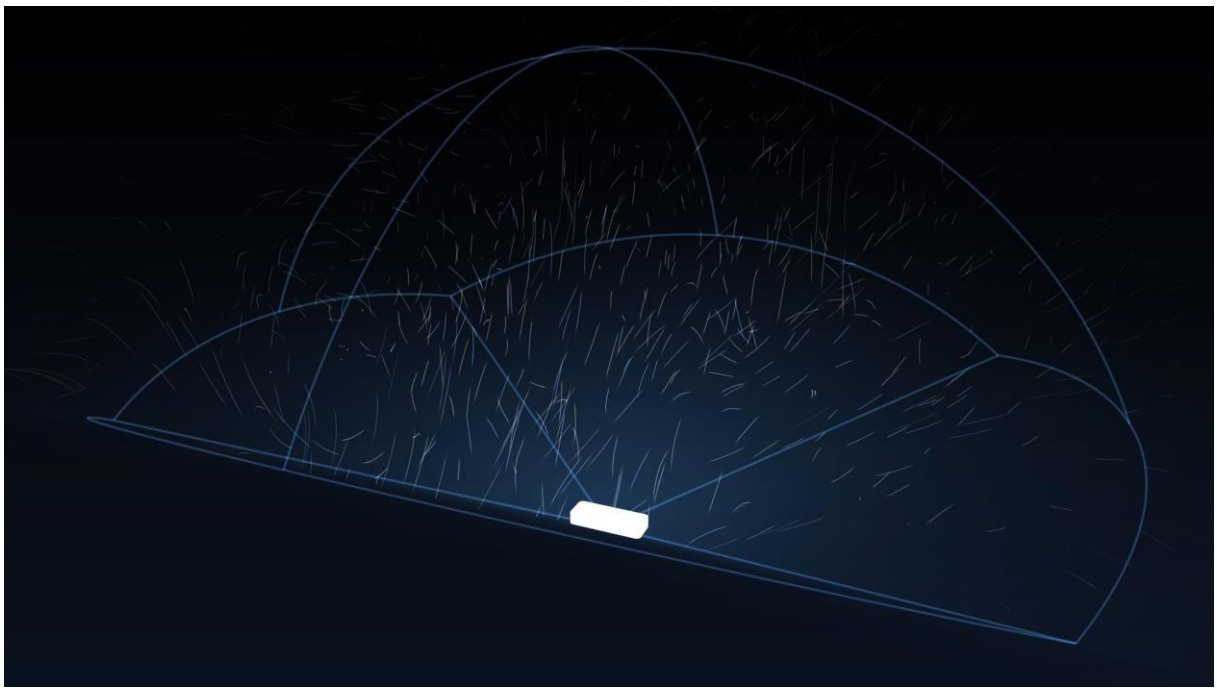
Još od samih početaka Weba krenulo se s imitiranjem funkcionalnosti aplikacija koje su do tada postojale samo na stolnim i prijenosnim računalima. Takve aplikacije su, u teoriji, lišene svake platformske ovisnosti, a korisnicima im na jednostavan način pristupaju sa svih strana svijeta te bilo kojega računala sposobnoga koristiti Web. Upravo mogućnost uniformnoga korištenja aplikacije neovisno o platformi lokaciji korisnika te sve veća prisutnost i razvoj mobilnih uređaja razlog su sve većoj popularnosti ovakvih aplikacija. Današnjim brzim tehnološkim razvojem to nije više samo pusta želja ili cilj, nego stvarnost.

Razvijene tehnologije nisu zaobišle ni područje računalne grafike pa je sada moguće raditi s 2D i 3D objektima unutar internetskoga preglednika korištenjem sučelja WebGL. Moguće je pisati aplikacije u jeziku JavaScript koje će korisnici pokretati na svojim uređajima. Iako trenutno brzina i kvaliteta ovako razvijenih aplikacija ne mogu konkurirati nativnim aplikacijama, daljnji razvoj dovest će vrlo vjerojatno do daljnjih poboljšanja.

U kontekstu ovoga projekta, uniformnost pristupanja aplikaciji i njezino jednostavno korištenje pomoću bilo kojega uređaja s pristupom Internetu bili su najveći argumenti za korištenje WebGL rješenja.

2.1 Uređaj Leap motion

Leap Motion kontroler je uređaj za detekciju položaja i pokreta prstiju ruke i sličnih objekata, koji je postao dostupan javnosti u siječnju 2013. godine. Njegova posebnost je u niskoj cijeni, malenim dimenzijama, brzoj i relativno kvalitetnoj detekciji informacija, jednostavnoj integraciji u postojeće aplikacije. Leap nije zamišljen kako bi u potpunosti zamijenio korištenje tipkovnice i miša, već kako bi radio s njima i tako poboljšao korisnikovo iskustvo i kreativnost. Budući da je njegov rad apsolutno neovisan o prisutnosti tipkovnice i miša, idealan je za uvjete u kojima je nepoželjan ili nemoguć redoviti kontakt s tipkovnicom i mišem, kao na primjer industrijska postrojenja. Jedini uvjeti za rad su spojenost na računalo te dovoljna svjetlost kako bi detekcija prstiju bila moguća. Uređaj Leap Motion radi na principu skeniranja prostora iznad senzora postavljenog na ravnu površinu, te detektiranja refleksije infracrvenih zraka od objekata u koje su one odaslane (Slika 1). Na boljim računalima, Leap pokazuje zavidne performanse praćenja pokreta brzinom od 200 okvira iscrtavanja u sekundi, preciznošću od jedne stotinke milimetra. Uređaj se trenutačno može koristiti na Windows, Mac i Linux operacijskim sustavima, te se aplikacije mogu razvijati u jezicima C++, C#, Objective-C, Java, JavaScript i Python.



Slika 2.1 Grafički prikaz vidnog polja Leap Motion kontrolera [7]

2.1.1 Dostupni podaci

U uvodnom dijelu je spomenuto kako uređaj skenira prostor i pruža podatke u strukturama čija promjena vrijednosti se mjeri u stotinama u sekundi. Struktura koja obuhvaća sve podatke dostupne od jednog skeniranja uređajem naziva se *Frame* (okvir). Glavna klasa svih podataka uređaja, poput položaja, volumena vidljivog uređajem, skaliranja i jedinica prostora, sadrži i povijest svih struktura okvira od pokretanja aplikacije (s, naravno, ograničenjem radne memorije). Jedan okvir sastoji se od slijedećih podataka:

Hands - strukture podataka položaja ruke. Postoje tri strukture ruke, jedna koja je prepoznata kao lijeva, druga kao desna ruka, a treća je ruka koja sadrži podatke za „neprepoznate“, nedovoljno i neispravno detektirane objekte. Primjer toga je detekcija samo 8 prstiju s dvije ruke korisnika, gdje će podatci preostala 2 prsta biti pohranjeni pod treću ruku, koja se zapravo, kako se može primijetiti, koristi kao spremnik referenci na memoriju trenutno nevažećih podataka, kako bi se poboljšale performanse i uklonile potrebe za konstantnim realociranjem memorije.

Pointables – „pokazivači“ - svi prsti i objekti koji se mogu klasificirati kao strukturom sličnim prstima (poput olovke ili štapića). Postoje 3 skupine *Pointable* objekata, za sve objekte koji su detektirani za neku od ruku u trenutačnom okviru izvršavanja aplikacije.

Fingers - prsti – ovo je najčešće korištena skupina *Pointable* objekata, koja sadrži kolekciju struktura s podatcima o detektiranom prstu.

Gestures – geste – spremnik za sve geste detektirane ugrađenim kodom Leap uređaja, uključujući geste koje su upravo detektirane (započete), završene, ili koje su usred detektiranja izvođenja.

S obzirom da Leap analizira cjelokupni pokret svih detektiranih objekata, moguće je dobiti i te podatke. Na primjer, ukoliko korisnik pomakne obje ruke u lijevu stranu, transformacija u strukturi okvira sadrži translaciju. Ako se obje ruke okrenu, okvir sadrži rotaciju, a ako se ruke približe ili udalje, detektirana je translacija. Leap koristi sve objekte koji su ispravno detektirani tijekom izračuna, te se sve transformacije mogu postići samo jednom rukom, ukoliko je samo ta ruka vidljiva

uređaju, no navedeni podatci su dostupni i za svaku ruku zasebno.

Struktura ruke ima vlastite atribute, navedene u nastavku:

PalmPosition – položaj dlana – centar dlana mjeren u milimetrima stvarnog prostora od zadanog središta Leap uređaja (inicijalno je to centar samog uređaja, no može se promijeniti na proizvoljnu točku)

PalmVelocity – brzina dlana mjerena u milimetrima stvarnog prostora, čije se skaliranje za pretvorbu u prostor Unity scene može proizvoljno promijeniti

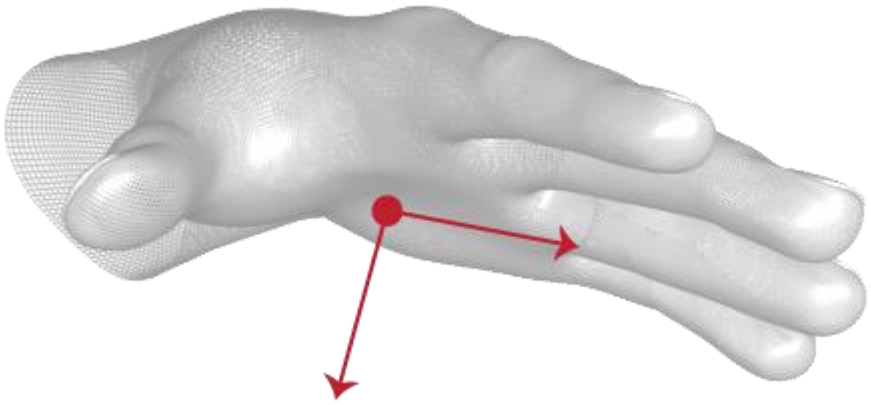
PalmNormal – smjer normale dlana – normalizirani vektor okomit na dlan ruke, u smjeru prostora ispod dlana

Direction – normalizirani vektor koji pokazuje u smjeru od dlana prema prstima

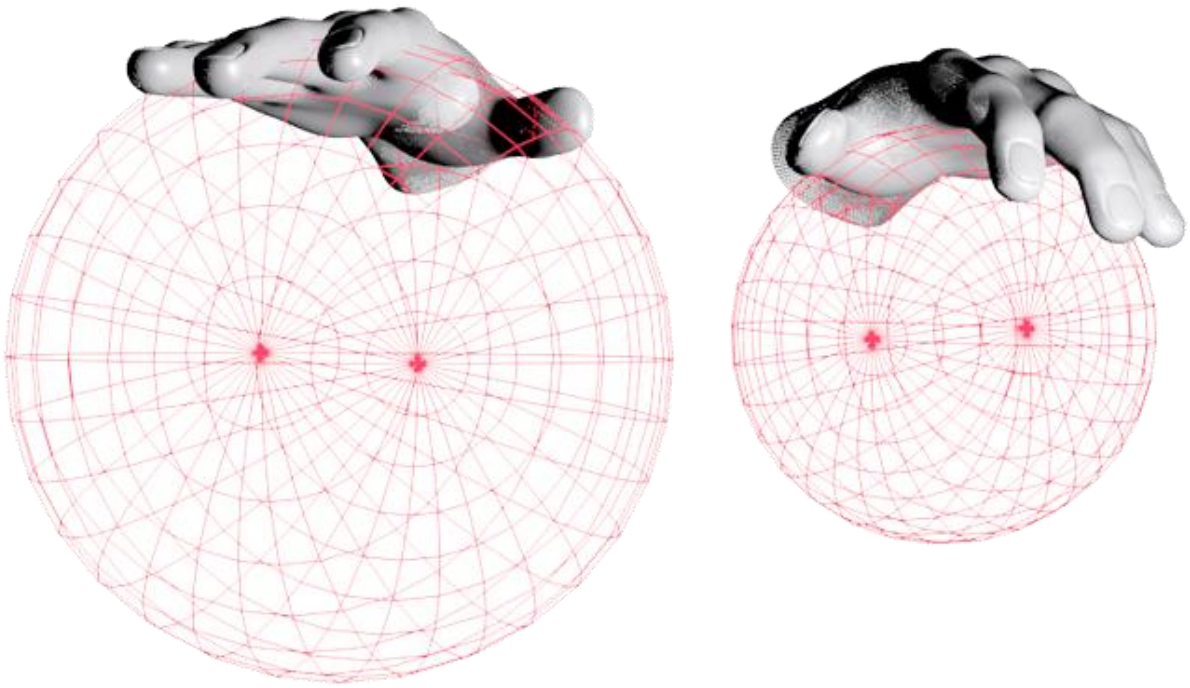
SphereCenter – centar kugle koja stane između prstiju i dlana, s obzirom na trenutnu zakrivljenost dlana i prstiju

SphereRadius - radijus kugle koja stane između prstiju i dlana, s obzirom na trenutnu zakrivljenost dlana i prstiju

Slika 2 pokazuje položaje vektora *PalmNormal* i *Direction*, a slika 3 pokazuje vizualizaciju kugle koja stane u ruku, te se može rekonstruirati iz podataka *SphereCenter* i *SphereRadius*.



Slika 2.2 Prikaz vektora PalmNormal i Direction za jednu od detektiranih ruku [16]



Slika 2.3 Prikaz ovisnosti položaja i radijusa kugle dostupne iz podataka ruke u ovisnosti o zakrivljenosti prstiju ruke [16]

Svi detektirani objekti „pokazivači“ imaju slijedeće podatke:

- *Length* – duljina vidljivog dijela objekta (od točke prekida spoja s odgovarajućom rukom do vrha)
- *Width* – širina vidljivog dijela objekta
- *Direction* – normalizirani vektor od baze do vrha objekta
- *TipPosition* – položaj vrha objekta mjeren u milimetrima stvarnog prostora od zadanog središta Leap uređaja
- *TipVelocity* – brzina vrha objekta mjeren u milimetrima stvarnog prostora po sekundi

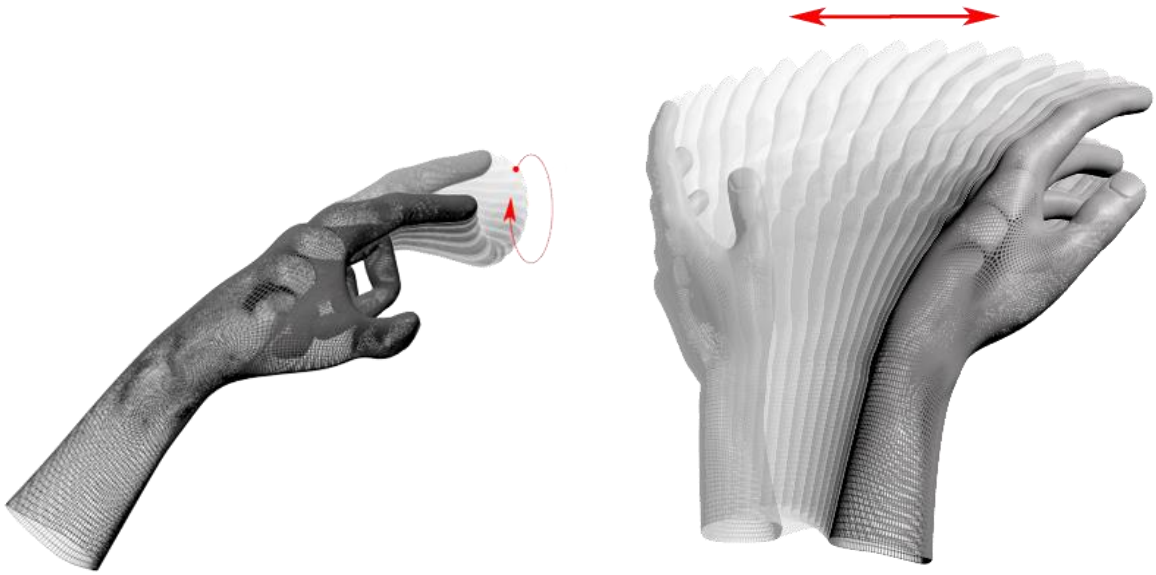
Navedeni podaci su jedini potrebni u vlastitoj implementaciji za detekciju gesti u ovom radu. Slika 4 prikazuje podatke *TipPosition* i *Direction* za prste jedne ruke.

Slika 2.4 Prikaz podataka TipPosition i Direction za prste jedne ruke [16]

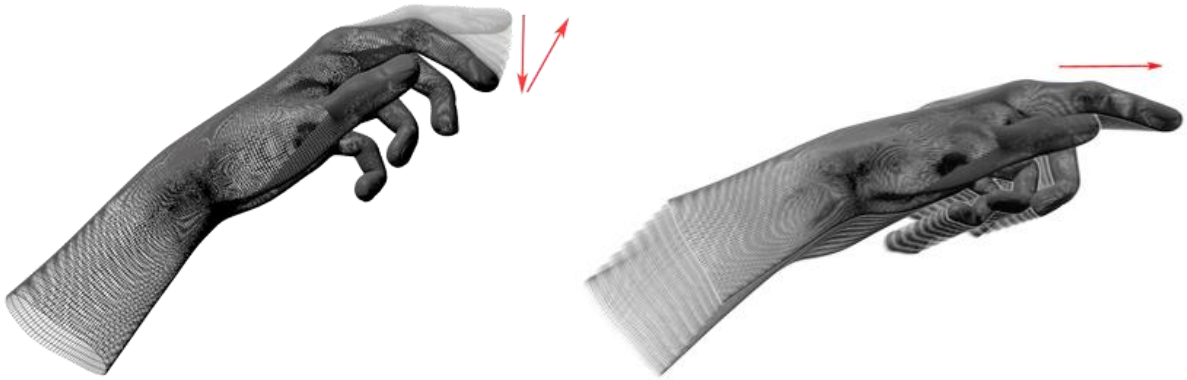
Postoje četiri vrste gesti čija detekcija je ugrađena u Leap uređaj, te se podaci o njima pridružuju svim podacima o objektima u strukturi svakog okvira postojanja gesti. U nastavku je dan pregled tih gesti:

- *Circle* - kružni pokret s jednim ispruženim prstom jedne ruke
- *Swipe* – linearni pokret ruke
- *Key Tap* – brzi pokret prstom dolje – gore, kao pritisak tipke na tipkovnici
- *Screen Tap* – brzi pokret prstom naprijed – nazad, kao dodir ekrana računala

Slike 5 – 8 prikazuju postizanje navedenih gesti, čime završava pregled dostupnih podataka iz uređaja Leap na temelju detekcija objekata stvarnog svijeta u jednom od okvira izvođenja aplikacije.



Slika 2.5 Prikaz postizanja gesti Circle (lijevo) i Swipe (desno) [16]

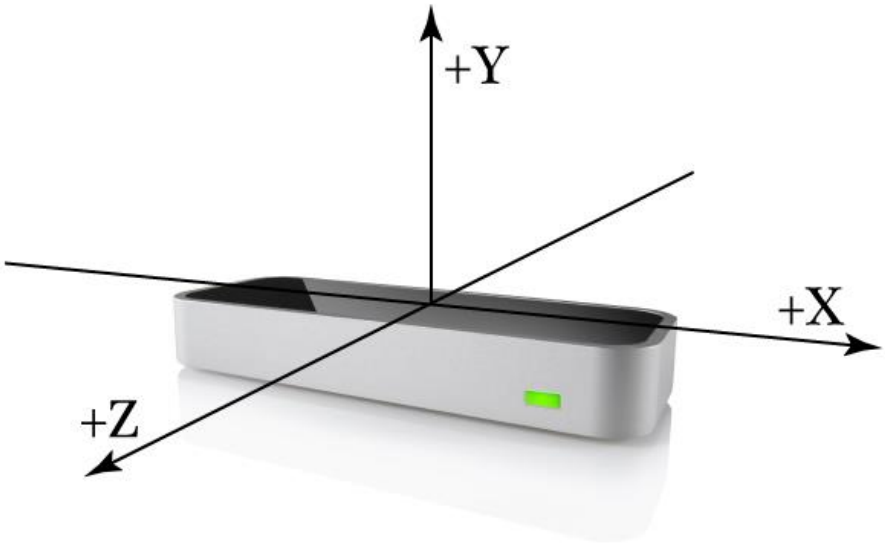


Slika 2.6 Prikaz postizanja gesti Key Tap (lijevo) i Screen Tap (desno) [16]

2.1.2 Uređaj Leap u Unity Engine okruženju

Za korištenje Leap uređaja u Unity okruženju dovoljno je uključiti prekompajlirane biblioteke dostupne na stranicama proizvođača uređaja, te skripti napisanih specifično za komunikaciju s podatcima uređaja u komponentno orijentiranom sustavu koji se koristi u tom okruženju. U ovom okruženju moguće je koristiti jezike C# i JavaScript, zasebno i istovremeno u različitim skriptama.

Koordinatni sustav u Unity-ju je lijevi, a Leap uređaja desni Kartezijev koordinatni sustav, kao što je moguće vidjeti na Slici 91. Centar sustava je postavljen na položaj samog uređaja, a skripte omogućavaju da se on proizvoljno pomakne, uz skaliranje odnosa prostora uređaja i prostora u Unity sceni.



Slika 2.7 Prikaz koordinatnog sustava uređaja Leap [16]

2.1.2.1 Cilj korištenja Leap uređaja

Cilj ovog projekta je, kao što je prije navedeno, isprobavanje korištenja različitih tehnologija s obzirom na različit način interakcije korisnika s aplikacijom na računalima s različitim operacijskim sustavima i drukčijim performansama. Kao jedno od zahtjevnijih mogućnosti s obzirom na snagu računala, Leap motion je nositelj tehnike bazirane na detekciji pokreta uz specijalizaciju za detekciju prstiju i ruke. S obzirom da je osmišljen za mogućnost upravljanja aplikacijama na temelju gesti, potrebno je osmisliti koje su akcije potrebne u ostvarivanju funkcionalnih zahtjeva projekta, te njihovo preslikavanje na pokrete korisnika. Nakon danog pregleda dostupnih podataka Leap uređaja kroz jedan okvir detekcije i analize podataka, postaje očito da će sve geste morati zadovoljavati mjeru različitosti od drugih izabranih gesta, tako da korisnik ne bi mogao slučajno izazvati neke akcije nepoželjne u tom trenutku pokretima ruke koji se neznatno razlikuju od pokreta potrebnih za željenu akciju.

U prvom koraku nude se dvije mogućnosti: korištenje ugrađenih gesti i implementacija vlastitih. Nakon kratkog ispitivanja ugrađenih gesti dolazi se do zaključka da su one nedovoljne brojnosti, raznovrsnosti i kvalitete za postizanje svih

akcija iz uvjeta projekta. Konkretnije, željene akcije su slijedeće: orijentacija modela i manipulacija pogleda na model, kontrola animacija u smislu pokretanje i zaustavljanje animacije, pokretanje animacije unazad, ili trenutne animacije ponovno, pomak na prethodnu i slijedeću animaciju, te kontrola zamjene između scenarija korištenja ciljnog uređaja.

2.1.3 Uređaj Leap u web okruženju

Leap skripta se u web aplikacije uključuje jednostavno kao i svaka druga JavaScript skripta. Može se spremiti lokalno s aplikacijom ili se može ubaciti veza na js.leapmotion.com kako se ne bi trebalo voditi računa o novijim verzijama.

```
<script src="//js.leapmotion.com/0.2.0/leap.min.js"></script>
```

I to je sve što je potrebno kako bi se moglo početi raditi s Leap Motion uređajem.

2.1.3.1 Korištenje uređaja

Jednostavnim instanciranjem objekta kontrolera omogućuje se dalji rad. Preko kontrolera primaju se informacije koje uređaj bilježi i reagira na događaje.

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

Kontroler nudi nekoliko tipova događaja. Kada se LeapJS spoji na *websocket* server poziva se `connect` događaj. Nakon toga se odabire protokol, okida se `ready` događaj. Kada se primi prvi okvir, okida se `deviceConnected` događaj da je uređaj spojen na računalo. Drugi tipovi događaja su:

- `connect` – klijent je spojen na *websocket* server
- `protocol` – odabran je protokol za spajanje
- `ready` - odabran je protokol
- `disconnect` – klijent se odvoji od *websocketa*
- `blur` – preglednik nije fokusiran
- `deviceConnected` – Leap uređaj je spojen na računalo
- `deviceDisconnected` – Leap uređaj nije spojen na računalo
- `frame` – kontorler je na kraju procesiranja okvira
- `animationFrame` - okvir koji se emitira u petlji animacije

I kako to zapravo izgleda:

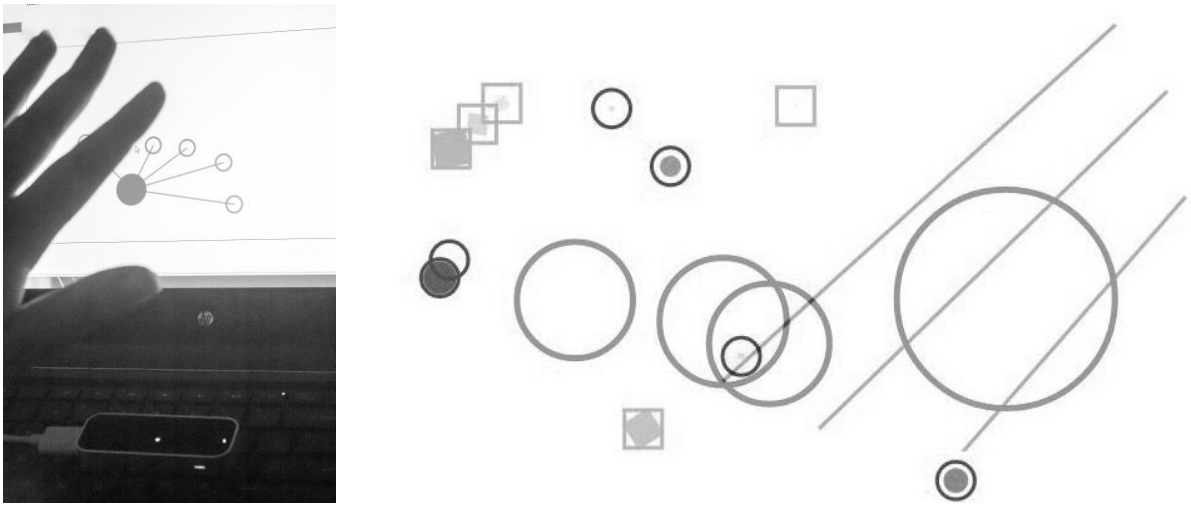
```
controller.on('connect', function() {
  console.log("Successfully connected.");
});
controller.on('deviceConnected', function() {
  console.log("A Leap device has been connected.");
});
controller.on('deviceDisconnected', function() {
  console.log("A Leap device has been disconnected.");
});
controller.connect();
```

2.1.3.2 Događaj okvira

Događaj okvira (eng. *frame event*) se poziva kao i drugi događaji. Glavna razlika je što vraća podatke okvira koji se dalje koriste za rad s informacijama koje nudi Leap uređaj.

```
controller.on( 'frame' , function( frame ) {
    // Frame code goes here
});
```

Neke od osnovnih stvari koje se mogu napraviti s informacijama su brojanje prstiju, vizualizacija ruke u virtualnom prostoru (vizualizacija interakcije), primjena ugrađenih gesti i još mnogo toga.



Slika 2.8 Vizualizacija ruke (lijevo) i ilustracija gesti: *swipe, circle, screen tap* (desno)

2.2 Glasovne naredbe

Sve više uređaja i aplikacija nudi mogućnosti korištenja glasovnih naredbi za rad. U sklopu ovog projekta proučili smo korištenje glasovnih naredbi uz korištenje Unity3D program za izradu računalnih igra i web aplikacija u web preglednicima.

2.2.1 Upravljanje animacijom glasovnim naredbama u Unity3Du

Animacijom je moguće upravljati i glasovnim naredbama. Zvuk koji dolazi s mikrofona obrađuje se i na taj način se određuje koja animacija će se izvršiti. Obradu zvuka moguće je izvršiti na nekoliko načina, od jednostavnijih i ograničenih do kompliciranih i funkcionalnih.

Zvuk se u Unity3Du predstavlja AudioSource komponentom, odnosno izvorom zvuka na objektu. Za potrebe snimanja i korištenja zvuka s mikrofona dovoljno je napraviti prazni objekt na kojemu se nalazi upravo ta komponenta i sve potrebne skripte za obavljanje analize.

Podaci iz mikrofona uzimaju se malim intervalima, a iz njih se uprosječivanjem računa glasnoća i korištenjem diskretne Fourierove transformacije frekvencija. Analizom ovih značajki moguće je pokretati animaciju.

Obradom vrijednosti glasnoće jednostavno se može odrediti prag za pokretanje animacije, no takav prag je statičan što je negativno jer ga šum može smesti i dopušta samo jednu naredbu (ako je glasnoća veća od dopuštene) zbog nepreciznosti mjerenja [10]. Upravljanje različitim glasnoćama u govoru ionako nije jako poželjna karakteristika, pa se signal dalje analizira do razine frekvencija.

Preko brze Fourierove transformacije (funkcije koja je ugrađena u Unity i mnoge druge sustave) moguće je dobiti frekvencijski spektar signala zvuka [11]. Daljnjom analizom tih podataka moguće je izolirati najzastupljeniju frekvenciju, pronaći šum ili dubinu govora. Ovi podaci moraju se uzimati iz kratkih, ali ne prekratkih zapisa (oko 100 ms) kako bi mogli uhvatiti neki relativno jednoličan interval zvuka, a da informacija koju time dobivamo nije beznačajna. Naime, ukoliko je interval prekratak obrada bi mogla potrajati i davati jednolične rezultate, a ukoliko je predug mogli bi pogreškom zanemariti neke utjecajne frekvencije. Rezultat

transformacije dat će nam najčešće najutjecajniju frekvenciju u nekom vremenskom intervalu iz čega bismo mogli odrediti npr. visinu glazbene note ili neku sličnu jednostavnu karakteristiku glasa. Mnogi postupci prepoznavanja govora zasnivaju se na obradi rezultata dobivenih Fourierovom transformacijom.

2.2.1.1 Implementacija

Nažalost, za dobivanje preciznih podataka preko Fourierove transformacije potrebno je napraviti cijeli sustav za prepoznavanje govora, čiji opseg nadilazi naše mogućnosti. Srećom, postoje mnogi već implementirani sustavi za prepoznavanje govora. Odabrani sustav ovdje je *Speech Recognition* koji se koristi na novijim Windows operativnim sustavima. Njegova slabost je u tome što se ne može preseliti na druge sustave. No, moguće je implementirati mrežnu verziju kroz koju na neko drugo računalo (s Windowsima) šaljemo podatke za obradu, a rezultat se vraća istom vezom. Upravo jedan takav program [12] koristimo za obradu podataka. Zbog toga što se Unity scene mogu sastaviti za veliki broj platformi, ovakva aplikacija, uz manje modifikacije, mogla bi raditi i na drugim sustavima ukoliko se uspostavi veza s nekim računalom na kojemu je pokrenut server za prepoznavanje govora. Sve ovo je naravno moguće napraviti i preko nekog drugog APIja sa sličnim funkcijama.

Za izvedbu na ovaj način potrebno je istovremeno pokretati server za obradu podataka i Unity scenu. Trenutna izvedba na serveru obrađuje zvuk te rezultate (jednu riječ iz gramatike) šalje na jedan prazni objekt u Unityju, gdje se preko tog rezultata upravlja animacijama. Gramatika je definirana preko liste riječi koje se mogu prepoznati (odnosno, prilikom obrade podataka traže se samo te riječi). Sam Speech Recognition sposoban je prepoznati riječi iz jednog od sedam najrasprostranjenijih jezika ukoliko je to jezik instalacije Windowsa na kojoj se nalazi. Potrebno je definirati i parametre za mrežnu komunikaciju (ukoliko su server i Unity klijent na istom računalu ovo se jednostavno namješta). Ukoliko su server i klijent na istom računalu brže je i praktičnije napraviti komunikaciju porukama između procesa, no ovaj primjer ima veću prenosivost i općenitiji je. Rezultat obrade je neka riječ iz gramatike. Skripta koja predstavlja klasu za tumačenje rezultata i pokretanje animacije uzima rezultat od komponente koja komunicira sa serverom i ovisno o dobivenoj riječi vrši akcije. Te akcije mogu biti animiranje slijedećeg koraka, animiranje prethodnog koraka,

završetak programa, promjena scenarija i druge, a nove operacije se jednostavno nadodaju – dodatkom nove riječi u gramatiku i nove opcije u switch naredbu koja bira akciju na osnovi detektirane riječi.

Sam odnos podataka i toka server-klijent može biti uspostavljen i jednostavnije, tako da se primanje podataka obavlja direktno na serveru. Na ovaj način jedini podaci koji se šalju mrežom su rezultati – jedan niz znakova po naredbi. Također, na objektu u Unityju ne treba postojati AudioSource komponenta, što ponešto ubrzava raspoznavanje.

Ovaj sustav može se unaprijediti na brojne načine. Iako je Windows *Speech Recognition* među najboljima [14], moguće je odabrati neki sustav koji radi na više platformi. Također je moguće smjestiti cijeli kod u Unity, no u tom slučaju on će raditi samo na novijim verzijama Windowsa. Iako bi to zahtijevalo ekstenzivan rad, u Unityju je moguće napraviti vlastitu obradu zvuka i prepoznavanje govora preko postupaka opisanih u mnogobrojnim radovima [15] koji nisu pretjerano kompliciranu i mogu raditi za jednostavne primjere.

2.2.2 Upravljanje animacijom glasovnim naredbama u WebGLu

Novi JavaScript *Web Speech API* omogućuje jednostavno korištenje raspoznavanja govora u web aplikacijama. Prednost korištenja WebGLa za grafičke aplikacije je što nije potrebna nikakva instalacija dodatnih priključaka, no za sada je *Web Speech API* dostupan samo u web pregledniku Chrome 25 verziji ili kasnijoj.

2.2.2.1 **Korištenje *Web Speech APIa***

Prvo je potrebno provjeriti da li preglednik podržava . *Web Speech API* tako da se provjeri postoji li `webkitSpeechRecognition` objekt, te ako postoji kreira se `webkitSpeechRecognition` objekt koji osigurava sučelje za govor, njegove atribute i rukovatelje događaja.

```
1. if (!('webkitSpeechRecognition' in window)) {
2.     upgrade();
3. } else {
4.     var recognition = new webkitSpeechRecognition();
5.     recognition.continuous = true;
6.     recognition.interimResults = true;
```

```
7.     recognition.onstart = function() { ... }
8.     recognition.onresult = function(event) { ... }
9.     recognition.onerror = function(event) { ... }
10.    recognition.onend = function() { ... }
11. ...
```

Zadana vrijednost za kontinuirano praćenje govora (atribut `continuous`) je postavljena na *'false'*, što znači kada korisnik prestane govoriti, raspoznavanje govora će završiti. Ovaj način je koristan za kratke ulaze teksta kao kratke ulazne stavke u formularima, no nije korisno za upravljanje aplikacijama pa ga postavljamo na *'true'* kako bi raspoznavanje nastavilo i kada korisnik napravi kratku pauzu u govoru.

Isto tako zadana vrijednost za `interimResults` je *'false'* što označava da ono što vraća raspoznavatelj govora je završno (eng. *final*) i neće se mijenjati. Ako se postavi na *'true'* mogu se dobiti rani, međurezultati koji se mogu promijeniti.

API podržava brojne jezike, a ako jezik nije zadan, uzima jezik korijenskog elementa HTML dokumenta.

Nakon što je postavljen jezik, poziva se `recognition.start()` za aktiviranje raspoznavanja govora. Kada se započne primati audio ulaz, poziva se događaj `onstart` nakon čega se za svaki novi set rezultata poziva `onresult` događaj.

```
1. recognition.onresult = function(event) {
2.   var interim_transcript = '';
3.   for(var i=event.resultIndex; i<event.results.length; ++i){
4.      if (event.results[i].isFinal) {
5.          final_transcript += event.results[i][0].transcript;
6.      } else {
7.          interim_transcript += event.results[i][0].transcript;
8.      }
9.   }
10. };
```

I na kraju rukovatelj događaja povezuje rezultate koje primi u dva stringa: `final_transcript` i `interim_transcript` s kojima aplikacija dalje rade.

2.2.2.2 Prednosti i nedostaci

Novi *Web Speech API* omogućuje korištenje glasovnih naredbi u web aplikacijama na jednostavan i vrlo dostupan način, te daje obećavajuće rezultate za specifikaciju u bliskoj budućnosti.

Web Speech API je podržan samo u web pregledniku Chrome 25 verziji ili kasnijoj, ali potpuna podrška nije implementirana unutar preglednika pa zahtijeva pristup Internetu, no kako se radi o web aplikacijama za koje se podrazumijeva spajanje preko Interneta, to i nije toliki nedostatak.

Najveći nedostatak je trenutna prevelika nepreciznost. Poboljšanja u preciznosti su moguća uz korištenje dodatnih algoritama za uspoređivanje prepoznate i ciljane riječi, no i tada se naredba često mora ponoviti (ponekad i više puta) kako bi se prepoznala kao ona ciljana kao sto se može vidjeti na primjeru:

Bez korištenja algoritma za usporedbu:	
Izgovorene riječi:	start, next, **continue**, **continue**, continue , **stop**, stop, down, **up**, **up**, **left**, **left**, up, right, left
Prepoznate riječi:	start, next, **continuum** , **continuum** , continue , **stock**, stop, down, **op**, **op**, **les**, **les**, up, right, left
Uz korištenje Levenshtein algoritma za usporedbu: [3]	
Izgovorene riječi:	down, up, left, right, down, down, up, left, right, next, next, start, continue, stop
Prepoznate riječi:	down, up, left, right, down, down, up, left, right, next, next, start, continue, stop

Tablica 2.1. Primjer prepoznavanja riječi pomoću *Web Speech APIa*

2.3 Dodirne naredbe

2.3.1 Windows 8 okruženje

Za razvoj Windows 8 aplikacije korišteno je Unity3D razvojno okruženje, što je povlačilo određena ograničenja pri razvoju zbog mogućnosti koje Unity3D ne nudi. Jedno od tih ograničenja je manjak podrške za višedodirne naredbe za Windows 8

aplikacije. Višedodirne naredbe bi se koristile za translaciju kamere. Gestom približavanja prstiju kamera bi se odmicala od modela a gestom udaljavanja prstiju jednog od drugog kamera bi se približavala modelu.

Tijekom razvoja ovog projekta nije pronađena besplatna alternativa koja bi omogućila unos višedodirnih naredbi za Windows 8 aplikaciju. Skripte koje daju te mogućnosti bile se dostupne putem Unity3D Asset Storea, ali uz plaćanje $40. Uz pomoć recenzija drugih kupaca uspostavilo se da kupovina nije vrijedna pa su se naredbe koje bi se obavljale uz više prstiju ostvarile unosom uz pomoć samo jednog prsta. Unos jednim prstom je istovjetan unosom uz pomoć miša.

2.3.2 WebGL okruženje

Web verzija aplikacije omogućava korisnicima odabiranje dijelova modela te pomicanje odabranih objekata, čime se omogućava jednostavno pregledavanje svih objekata. Kako bi ovo uopće bilo moguće, mora postojati način kojim bi se odredilo koji objekt je, među svim objektima u sceni, korisnik zbilja i odabrao.

2.3.2.1 Ubrzanje odabiranja objekata scene

Inicijalno rješenje odabiranja objekata temelji se na postupku praćenja zrake. Na temelju podatka o poziciji na kojoj se nalazio miš ili prst te informaciji o kameri moguće je konstruirati zraku koja prolazi kroz scenu. Ako zraka prolazi kroz neki objekt modela, tada je korisnik možda i odabrao promatrani objekt. Kako bismo bili potpuno sigurni, poredat ćemo probodene objekte na temelju udaljenosti probodišta od kamere te odabrati najbliži takav objekt.

Iako je ideja relativno jednostavna, problem se javlja kada je potrebno dohvatiti samo one objekte kroz koje zraka doista i prolazi. Naime, naš model ne može sam reći probada li ga zraka ili ne jer on nije ni svjestan naših namjera pa posao moramo obaviti sami.

Početno rješenje ovoga problema koristi najjednostavniji mogući postupak. Prolazeći kroz sve objekte u sceni te sve njihove poligone moguće je provjeriti probada li zraka neki od objekata te ako da – dovoljno je izdvojiti željene objekte te ih poredati prema udaljenosti od kamere.

Mana ovoga postupka jest što mora ispitati odnose svih objekata i kreirane zrake kako bi pronašao probodišta. Za modele s ne pretjerano velikim brojem objekata i poligona ovo ne predstavlja problem, brzina značajno pada s povećanjem složenosti modela, što može nepoželjno usporiti korištenje aplikacije. Naravno, usporavanje rada aplikacije sasvim uništava razlog zbog kojeg je odabiranje objekata i omogućeno – kako bi u realnom vremenu bilo moguće pregledavati dijelove izgrađenoga modela.

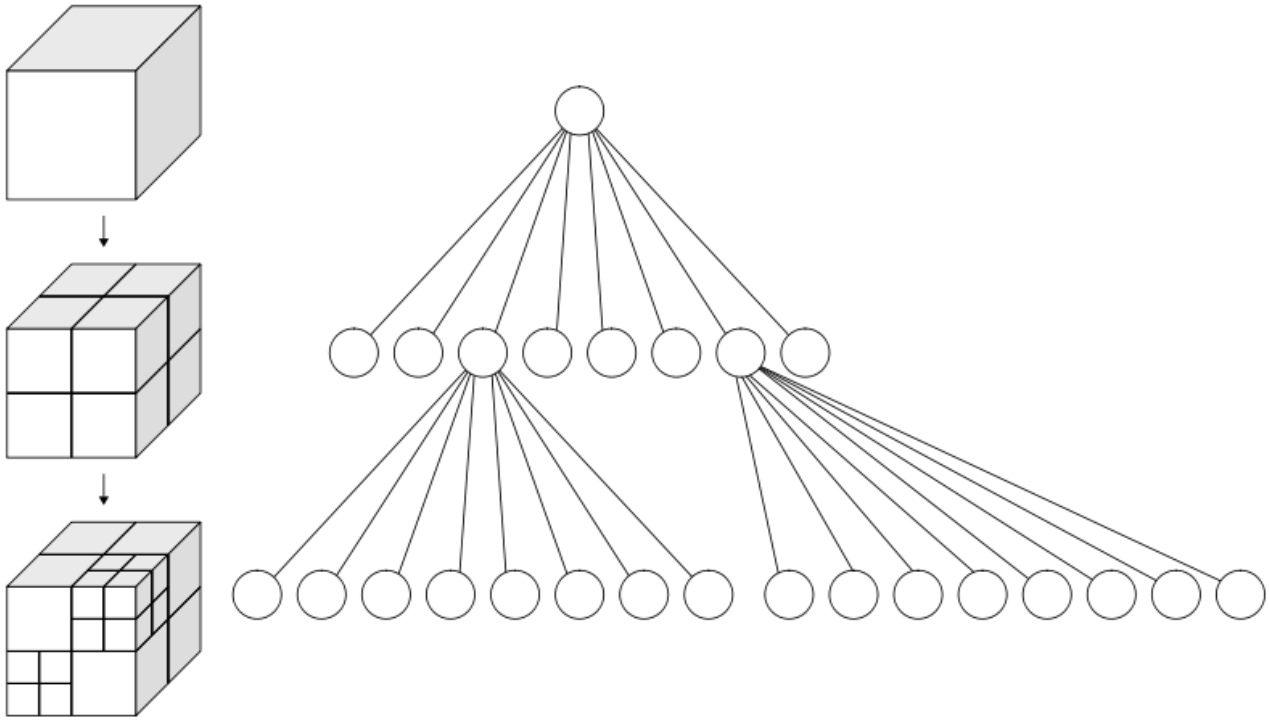
2.3.2.2 Oktalno stablo

Scenu koju želimo prikazati na ekranu moguće je uvijek na neki način organizirati, što je zapravo često i vrlo poželjno. Organizacija se često obavlja podjelom scene na nekoliko dijelova – bilo jednakih bilo različitih. Cilj takve podjele jest znati za svaki dio scene koji objekti se u njemu nalaze. Vrlo jednostavan primjer u kojem je važno takvo znanje jest detekcija sudara jer detekciju je potrebno provoditi samo za objekte koji se nalaze unutar istoga potprostora scene.

Upravo je podjela scene jedno od rješenja kojim se može ubrzati proces odabiranja objekata jer je problem vrlo sličan postupku otkrivanja sudara. Naime, u našem slučaju zraka se neće samo sudariti s objektom, nego će ga još i probosti, ali to i nije toliko važno.

Jedan od načina organizacije scene omogućavaju takozvana k-d-stabla, koja se koriste za organizaciju scene u k-dimenzionalnom prostoru. Budući da se aplikacija bavi 3D prostorom, koristi se jedna varijanta takvoga stabla nazvana oktalno stablo.

Oktalno stablo dijeli prostor scene u oktante – 8 jednakih kubičnih dijelova. Svaki od oktanata moguće je dalje rekurzivno dijeliti na manje potprostore kako bi se dobila finija podjela prostora. Odluka o tome hoće li se potprostor dalje dijeliti često se temelji na broju objekata koji se nalaze u oktantu pa se definira i maksimalni broj objekata koji mogu biti u istome oktantu te ako je trenutni broj objekata veći od nekog unaprijed zadanog broja, dio scene se dalje dijeli kako bi se omogućila preciznija podjela scene.



Slika 1. Prikaz rekurzivne podjele scene na oktante (slika s Wikipedije)

Slika 1 prikazuje način podjele scene te vezu između podjele i stabla. Inicijalna scena predstavlja korijenski čvor stabla. U sljedećoj razini nalazi se 8 čvorova i oni predstavljaju po jedan oktant dobiven podjelom originalne scene. Ako se podjela nastavi i dalje, svaki čvor može biti roditelj još 8 čvorova, što označava podjelu oktanta predstavljenog čvorom na još 8 dijelova.

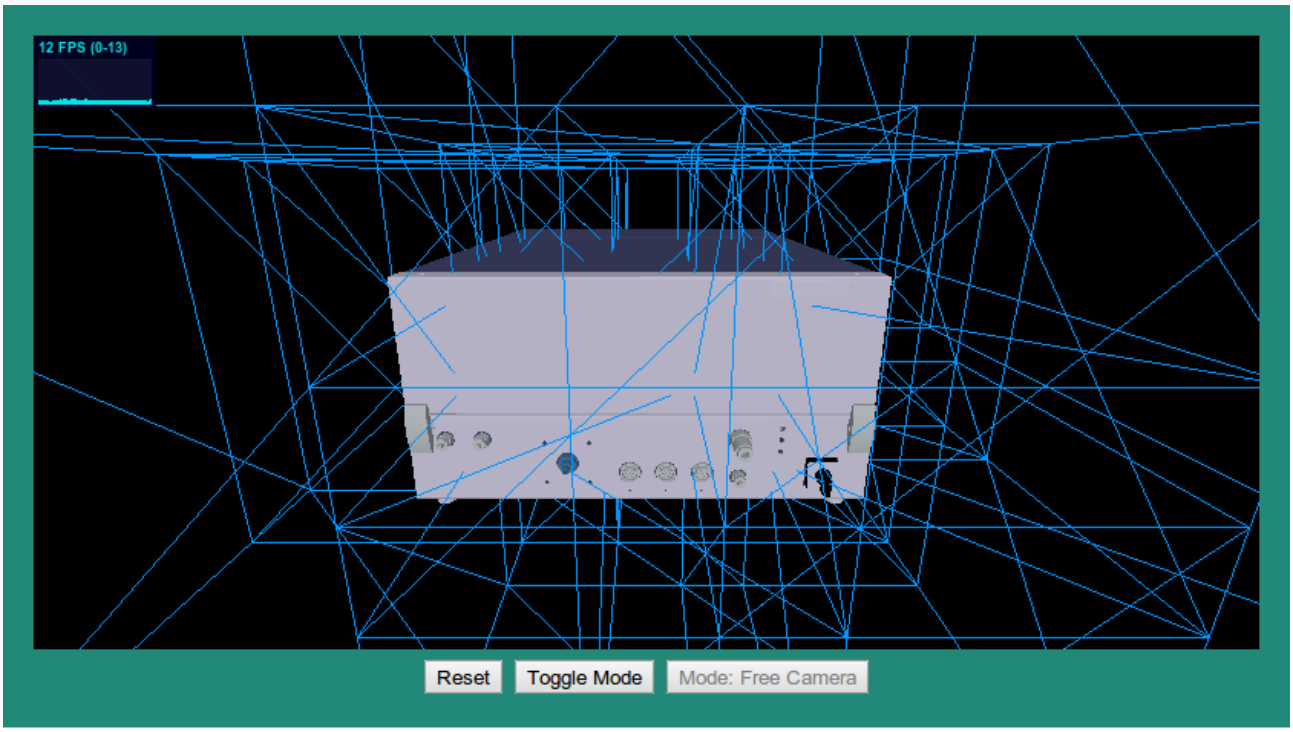
Logično je zapitati se što se točno postiže ovakvom podjelom scene u kontekstu problema koji se ovdje rješava. Podsjetimo se, krajnji cilj je podijeliti scenu na nekoliko dijelova što bi trebalo omogućiti brže pronalaženje probodenih objekata.

Oktalno stablo elegantno rješava problem podjele prostora jer originalnu scenu dijeli rekurzivno na oktante dok god pojedini oktant nije “prepunjen”, odnosno dok god sadrži “previše” objekata, gdje “previše” definiramo cjelobrojnom konstantom.

Još ostaje vidjeti kako ovakva podjela može pomoći u ubrzavanju procesa pretraživanja. Naime, želimo li pronaći koje objekte zraka probada, možemo pregledavati sve oktante te odabrati probodene objekte. Nažalost, to ne bi predstavljalo poboljšanje jer bi se opet obavljala iscrpna pretraga. Štoviše, proces bi bio još dugotrajniji jer potrebno je uzeti u obzir i vrijeme potrebno za izgradnju stabla.

Naravno da prihvatljivo rješenje doista postoji i ono je sljedeće: umjesto da se iscrpno pretražuju svi potprostori, potrebno je obaviti pretraživanje stabla. Stablo se pretražuje od korijena i slično je za svaki posjećeni čvor. U korijenu provjeri se kojem oktantu zraka pripada, što možemo učiniti jer je opisana pravcem. Kada se odabere pripadajući čvor u koji se postupak dalje grana, provjerava se je li pretraživanje došlo do lista stabla. Ako je, proces pretraživanja je gotov, a ako nije – nastavlja se s grananjem i pretraživanjem. Jednom kada postupak dođe do lista stabla, pronašli smo dio scene u kojem se nalaze samo oni objekti koje bi zraka mogla probosti. Kako bismo bili pronašli točno one objekte za koje to vrijedi, izvodi se iscrpno pretraživanje potprostora.

Smisleno je zapitati se koji je konačan dobitak provedenoga postupka ako se na kraju opet provodi iscrpna pretraga. Ako pažljivije promotrimo postupak, uočljivo je da svako grananje vodi do potprostora koji je 8 puta manji od prethodnoga. To pak znači da krajnji oktant sadrži značajno manje objekata pa iscrpno pretraživanje više ne oduzima toliko vremena.



Slika 2.9 Prikaz modela unutar aplikacije i izgrađenoga oktalnog stabla

2.3.2.2.1 Prednosti i nedostatci oktalnoga stabla

Organizacije scene općenito, a samim time i oktalno stablo, predstavlja potencijalan napredak ako je potrebno obavljati ispitivanja nad objektima scene jer smanjuju prostor pretraživanja algoritma, a samim time i ubrzavaju kompletan postupak. Samim time oktalno stablo pogodno je uklanjanje skrivenih linija i površina, detekciju sudara, traženje najbližih susjeda te slične postupke koji moraju obavljati pretraživanje scene poput traženja objekata koje je korisnik odabrao.

Problem je što generiranje stabla potencijalno usporava generiranje scene, pogotovo za veći broj objekata jer je potrebo podijeliti scenu te generirati stablo, koje već i za mali broj razina može imati veliki broj čvorova, što je računalno zahtjevan proces.

Dodatan problem kod korištenja oktalnoga stabla jest da je prilikom svake promjene scene potrebno osvježiti podatke jer objekti mogu prelaziti iz jednog oktanta u drugi pa podaci ne bi bili točno kada se stablo ne bi osvježavalo. To je također vremenski zahtjevno i kod jako velikih scena može predstavljati problem. Na sreću, model nad kojim se radilo nije pretjerano velik pa ovo nije bio problem.

2.3.2.3 Dodatan spremnik

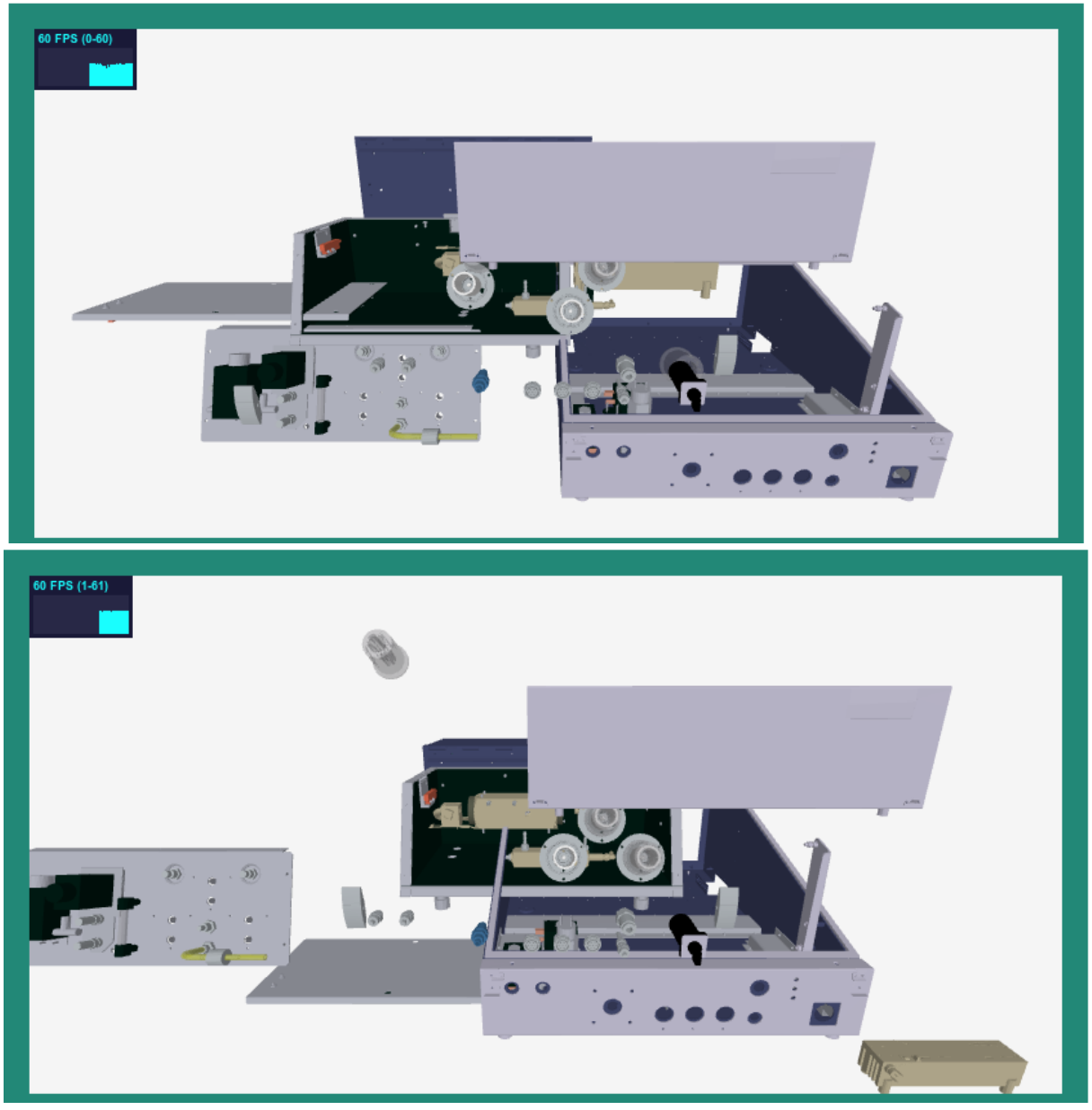
Zapitajmo se kako se uopće na ekranu dobije slika scene. Znamo da je scena u OpenGL sustavu definirana korištenjem svjetla i modela te postoji kamera koja određuje kamo se gleda. Kada želimo prikazati stanje scene na ekranu potrebno je tu scenu pretvoriti u dvodimenzionalni sustav. OpenGL se sam pobrine da su bliži objekti vidljivi dok oni koji su iza bližih objekata nisu vidljivi.

Ideja ovog pristupa je iskoristiti to ponašanje OpenGL sustava na sljedeći način. Svakom objektu dodijelit ćemo drugu boju koja će predstavljati neki identifikator. Potpuno je nebitno koja je to boja, ona ionako nikada neće biti prikazana korisniku. Ona nama služi samo za identifikaciju. Nakon što OpenGL obavi pretvaranje scene u dvodimenzionalni sustav, mi možemo spremnik pitati koja boja se nalazi na pritisnutoj lokaciji. Nakon toga možemo pronaći kojem objektu odgovara tražena boja. Potrebno je napomenuti da svjetla i slični efekti trebaju biti onemogućeni kako ne bi promijenili boju objekta.

2.3.2.3.1 Prednosti dodatnog spremnika

Metoda se izuzetno lako implementira i pokazuje dobre rezultate po pitanju brzine. Valja napomenuti da je vremensko trajanje više manje konstantno i za odabir objekta će nam potrošiti jednu sličicu. Primjerice, ako animacija ima 60 sličica u sekundi, odabirom jednog objekta će u tom trenutku biti 59 sličica u sekundi.

Sljedeće dvije slike pokazuju razliku u broju sličica po sekundi za staru i novu implementaciju.



Slika 2.10 Određivanje odabranog objekta pomoću iscrtavanja u dodatni spremnik

Na prvoj slici su jasno vidljivije oscilacije u broju sličica po sekundi zato što je dio procesorskog vremena otišao na provođenje algoritma praćenja zrake i kasnilo se sa slanjem objekata do grafičke kartice. U drugom slučaju tih oscilacija gotovo i nema. Objekti se konstantno šalju u grafičku karticu. Jedina stvar na koju se dodatno potroši procesorsko vrijeme je traženje kojem objektu pripada koja boja. Korištenjem algoritma poput raspršenog adresiranja to vrijeme postaje zanemarivo.

2.3.2.3.2 Nedostatci dodatnog spremnika

Valja napomenuti da ovaj pristup ipak ima nekih svojih ograničenja. Usko grlo postaje sabirnica između grafičke memorije i radne memorije računala. Naime, spremnik se nalazi u memoriji grafičke kartice i centralni procesor koji izvršava našu aplikaciju i reagira na korisničke podražaje, nema izravan pristup memoriji spremnika već samo pristup radnoj memoriji. Iz tog razloga potrebno je tražiti grafičku karticu da preko sabirnice te podatke prebaci u radnu memoriju kako bi im se moglo pristupiti. Taj problem dolazi do posebnog izražaja kod velikog broja odabira objekata. U tom slučaju je puno bolje implementirati dobru strukturu podataka poput oktalnog stabla koja drastično reducira broj provjera poligona.

Također moguće je imati samo konačan broj objekata, tj. broj objekata je izravno definiran brojem jedinstvenih boja i iznosi 16581374. Razlog zašto nedostaje jedna boja je zbog boje pozadine.

3. Opis razvijenog proizvoda

3.1 Izrada animacija

Sve animacije su rađene u besplatnom programu za modeliranje Blender. Korišten je dobiven .blend model Microsoot uređaja. Za potrebu izrade druge animacije, dodani su jednostavni objekti koji služe kao zamjene za cijevi, pločice i žice unutar uređaja. Animacije u Blenderu se izrađuju preko niza ključnih okvira, odnosno Key Frameova, a izradu istih ću opisati kroz nekoliko koraka.

Prvo je potrebno osigurati da se tijelo ponaša na odgovarajući način, pri čemu se uglavnom misli da ima dobru os postavljenu kao os oko koje se tijelo okreće. Ukoliko samo translatiramo predmet, to nije potrebno, ali u slučaju rotacije (kojih je bilo podosta tijekom izrade animacija) je nužno ako želimo da se tijelo ponaša na odgovarajući način. Najjednostavnije je odrediti os rotacije tako da se izabere jedan od bridova u Edit Modeu te se na njega pozicionira trodimenzionalni kursor Blendera, a zatim se u Object modeu odredi pozicija trenutnog kursora kao Origin point tijela.

Nakon što smo po potrebi odredili osi rotacije, potrebno je postaviti početan Key Frame te odabrati transformaciju koju ćemo vršiti. Blender nudi transformacije translacije, rotacije, skaliranja te kombinacije istih. Nakon što je određen početan Key Frame, odabiremo na vremenskoj crti gdje će završiti naša transformacija premještanjem kazaljke, a zatim primijenimo tu transformaciju na tijelo.

Po završetku toga, samo još jednom ponovimo postavljanje Key Framea te ćemo time dobiti transformaciju koju smo htjeli. Blender automatski interpolira između ta dva položaja tijela na sličicama između dva Key Framea. Ponavljanjem ovog postupka za sve potrebne objekte dolazimo do funkcionalne animacije.

3.1.1 Priprema modela

Priprema modela u Unityu se odvija u nekoliko jednostavnih koraka: Prvo je potrebno stvoriti novi projekt te u njega unijeti .fbx animaciju napravljenu pomoću Blendera na način opisan u prethodnom dijelu.

U novonapravljenom projektu potrebno je uvesti željene animacije kao nove assete u projektu te je potrebno uvesti nekoliko promjena u njihovim postavkama. Pod „Model“ je faktor skaliranja bilo potrebno postaviti na 0.1, a pod „Rig“ Animation Type postaviti na Legacy, a Generation na Store in Root (New).

Za kraj, potrebno je pod Animation razdvojiti razne dijelove animacije na kraće dijelove te im odrediti koje sličice obuhvaća koji dio animacije.

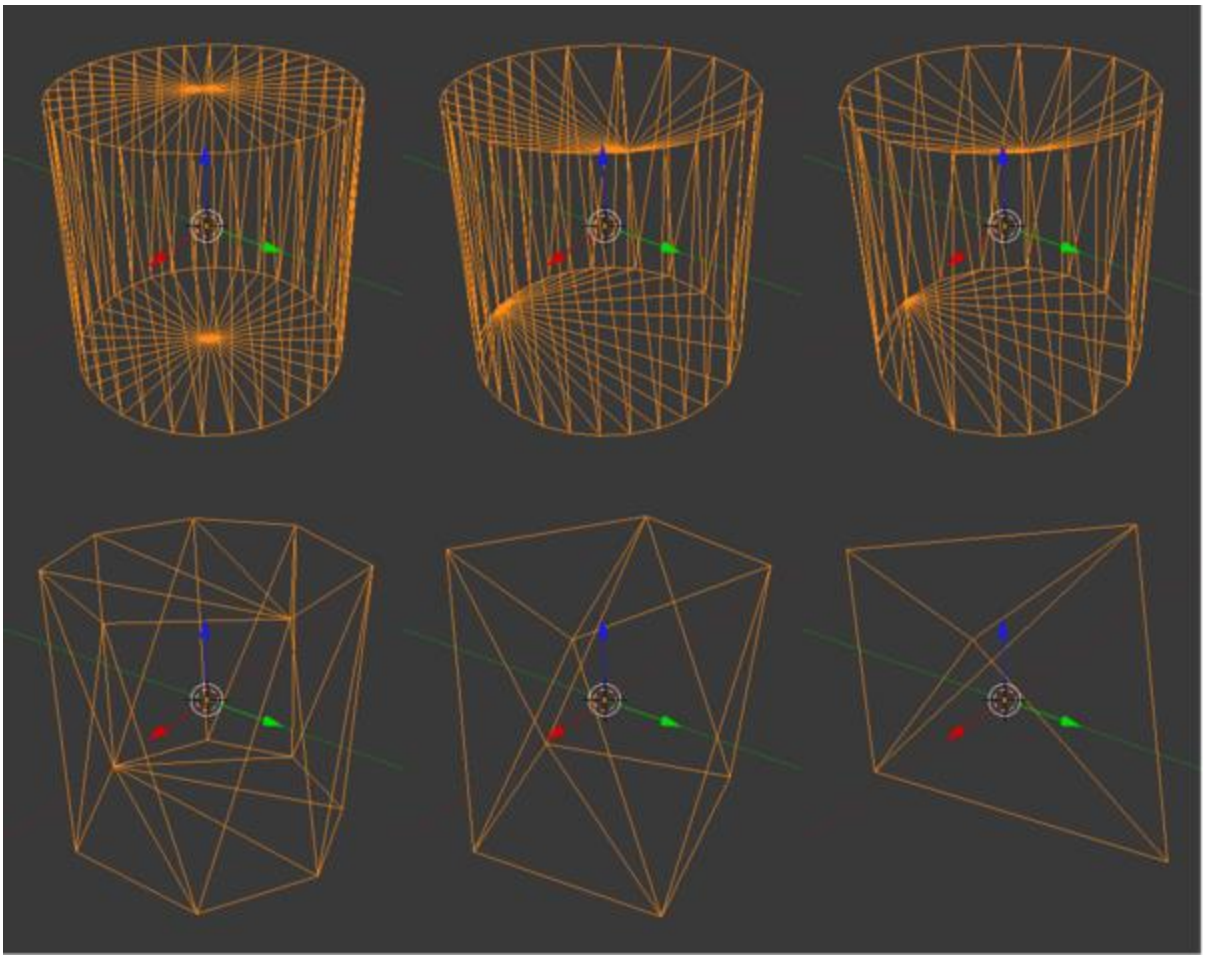
3.1.2 Pojednostavljenje modela

Model koji nam je prvotno dan na raspolaganje je imao određene probleme s performansama, posebno ukoliko bi se prezentirao unutar preglednika web stranica. Iz tog razloga, bilo je potrebno unijeti pojednostavljenja kako bi se optimirao rad programa.

Kao i za izradu animacija, za pojednostavljenje modela je također korišten besplatan program za modeliranje Blender. Blender u sebi sadrži operator Decimate koja olakšava izradu pojednostavljenog modela objekta.

Operator Decimate prolazi kroz vrhove ili poligone objekta te spaja one vrhove ili poligone za koje zaključi da neće dovesti do značajnije promjene izgleda objekta. Takvim smanjenjem broja poligona potrebno je manje memorije za zapis objekata te je ubrzano njihovo učitavanje te izvođenje nekih animacija.

Operator Decimate zahtjeva kao ulazni podatak postotak poligona koje želimo zadržati, ukoliko kao taj broj stavimo „100“ neće doći do nikakve promjene, a ukoliko stavimo „0“ uklonit će sve poligone. Potrebno je naći optimalan omjer kvalitete prikaza i performansi, a utjecaj različitih ulaznih vrijednosti na izgled objekta se može vidjeti u priloženoj slici. Za potrebe ovog projekta, uklonio sam 40% poligona, što nije dovelo do vidljive promjene, a veličina datoteke se smanjila za odgovarajućih 40tak posto.



Slika 3.1 Pojednostavljenje modela

.

3.2 Upute za korištenje

3.2.1 Unity3D okruženje

3.2.1.1 Uređaj Leap motion

Interakcija u smislu promjene orijentacije modela i pogleda kamere, kontrola animacije i scenarija je u potpunosti ostvarena u skladu s očekivanjima, uz mogućnosti naknadne promjene pridruživanja gesta na željene akcije, te promjene parametara pozicioniranja objekata i detekcije gesta.

S obzirom na isprobano lošu kvalitetu ugrađenih gesta Leap uređaja, implementiran je vlastiti mehanizam za detekciju definiranih gesta.

U ugrađenim gestama, svi detektirani objekti (prsti) su ulazili u obzir pri detekciji gesti. Leap uređaj ne može detektirati ispravno dva prsta iste ruke koji su spojeni jedan uz drugoga kao dva objekta, već ih detektira kao jedan. Ukoliko se pri pokretu jedan prst malo savije prema šaci, ili približi drugom, on će nestati iz detekcije iako se radilo o relativnom malenoj promjeni. Zbog navedenih problema, implementirano je vlastito filtriranje i detektiranih prstiju na one prikladne i neprikladne za korištenje u praćenju i detekciji gesta. Kriterij za prst prikladan za praćenje u svrhu detekcije gesta je slijedeći: prst mora biti dovoljno udaljen od referentnog centra za ovo dodatno odbacivanje prstiju. Ostvaren je izbor između dva centra za obje ruke i njima pripadajuće detektirane prste. Prvi centar je položaj zgloba ruke, a drugi je centar sfere koja se može tangencijalno smjestiti između prstiju ruke. Pokazalo se da je za prosječnu odraslu osobu ista vrijednost parametra dovoljna za ispravno odbacivanje neprikladnih prstiju. Nakon što je to ostvareno i testirano, krenula je implementacija vlastitih gesta.

Prva vrsta geste je promjena položaja, te se manifestira promjenom položaja X prstiju za barem Y jedinica prostora u Unity okruženju, gdje postoje različiti iznosi minimalnog pomaka za prvi prst koji se detektira pomak, te svih ostalih prstiju, koji se razmatraju uz manji pomak nakon detekcije prvog prsta, te se konačan broj prstiju uz pomak računa kao jedna ostvarena gesta.

Druga vrsta geste je detekcija određene brzine prstiju, gdje također postoje različiti parametri za prvi prst koji postigne tu brzinu, te za provjeru ostalih prstiju koji bi bili uključeni u ostvarenu gestu.

Obje vrste geste imaju nezavisni mehanizam praćenja, te na sceni Unity okruženja može biti prisutan neograničen broj sustava za njihovu detekciju. Za potrebe ovog projekta, u sceni je prisutan po jedan sustav za praćenje po tipu geste.

S obzirom na zahtjeve projekta, bilo je potrebno odlučiti kakve geste će pokretati koje od potrebnih akcija za interakciju s korisnikom. Eksperimentalno se pokazalo da geste s više od 5 prstiju nisu praktične, jer zahtijevaju više od jedne ruke, te maksimalnu preciznost korisnika. Za geste s četiri i pet prstiju pokazalo se da su problem sa strane detekcije Leap uređaja, te je konačna odluka da se akcije ograniče na geste koje se mogu postići korištenjem jednog, dva ili tri prsta.

Odabir konačnog pridruživanja gesti na akcije je slijedeći:

- Geste položaja dva prsta koriste se za orijentaciju modela.
- Geste brzine jednog prsta koriste se za kontrolu animacija modela
- Geste brzine tri prsta koriste se za kontrolu scenarija korištenja modela

Testiranje odabranih gesti u aplikaciji na više subjekata i uređaja pokazalo je da ostvaruju potrebne uvjete:

- jednostavnost – korisnici su lako shvatili i brzo memorizirali geste
- scenariji – sve tražene akcije za kontrolu modela, animacija i scenarija su ostvarene gestama
- preglednost – model i sve njegove promjene su jasno vidljive
- brzina – aplikacija funkcionira u razumnoj brzini, te su sve ispravne geste odmah prepoznate
- prenosivost – uz instalaciju potrebnih biblioteka, aplikacija je radila na svim isprobanim računalima unutar granica njihovih performansi

3.2.1.2 Glasovne naredbe

Aplikacija za upravljanje modelom glasovnim naredbama pokreće se pokretanjem servera za prepoznavanje govora i Unity izvršne datoteke. Alternativno,

moguće je pokrenuti izvršnu verziju u Unityju koja sama pretvara govor u riječi. U oba slučaja, riječi koje se prepoznaju uzimaju se iz baze riječi i njima se kasnije pridodaju naredbe. Zbog toga moguće je ubaciti velik broj riječi u gramatiku (a time i proširiti funkcionalnost), ali što ih je više to će se one slabije prepoznavati (za slične riječi postoji mogućnost detekcije krive koji proizlazi iz parametra praga za detekciju riječi).

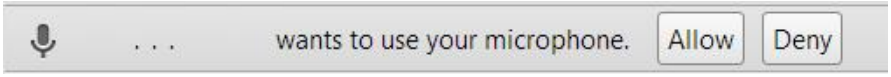
Robusnost sustava ovisi i o samim riječima koje su odabrane. Zbog svojstava Speech Recognizera odabrane riječi su engleske i to slijedeće: finish (izlazak iz programa), reset (postavljanje animacije na početak), switch (promjena na slijedeću animaciju), next (slijedeći korak trenutne animacije), reverse (prethodni korak trenutne animacije). Sustav za prepoznavanje govora vraća riječi koje se prikazuju na ekranu (kašnjenje do 1s) uz izvršavanje te naredbe. Riječi koje se ne prepoznaju nikako ne utječu na rad aplikacije.

3.2.2 Web okruženje

Web aplikacija demonstrira osnovne funkcionalnosti korištenja Leap motion uređaja i glasovnih naredbi. Sve funkcionalnosti implementirane u Unity okruženju su moguće i u web okruženju. U sklopu projekta, kada se aplikacijom kontrolira pomoću Leap uređaja, koriste se osnovne geste koje prepoznaje Leap.js biblioteka [8]. Ako se pređe na glasovne naredbe, kako bi se započelo raspoznavanje govora, korisnik mora pritisnuti na sliku mikrofona koji pokreće raspoznavanje govora.

Slika 3.2 Prikaz različitih stanja upotrebe Web Speech APIa: Čekanje na aktivaciju (lijevo), aktivan (sredina), zabranjeno korištenje (desno)

Chrome preglednik će pitati korisnika za dopuštenje korištenja mikrofona, te u slučaju potvrđivanja će izazvati pokretanje događaja `onstart`.



Slika 3.3 Dopuštenje korištenja mikrofona

Trenutna implementacija podržava generičke naredbe upravljanja animacija: pokreni, zaustavi, nastavi i rotiranje objekta: gore, dolje, lijevo, desno na engleskom i njemačkom jeziku.

3.2.3 Windows 8

Aplikacija za Windows 8 platformu može se koristiti kao aplikacija za stolna računala koja za interakciju koriste tipkovnicu i miš, te kao aplikacija koja za interakciju koristi unos preko ekrana osjetljivog na dodir (eng. *touchscreen*). Većina naredbi aplikacije zadaje se preko tipki na ekranu te se tako unosi pomoću miša i ekrana svode na isto.

Rotacija modela može se obavljati dvjema tipkama tipkovnice ili pak unosom dodirom prsta preko ekrana osjetljivog na dodir. Površina ekrana je podijeljena na devet dijelova. Dodirom središnjeg i sva četiri kutna dijela ne događa se ništa. Gornji središnji dio na dodir rotira model gore, desni središnji desno, lijevi središnji lijevo i donji središnji dolje.

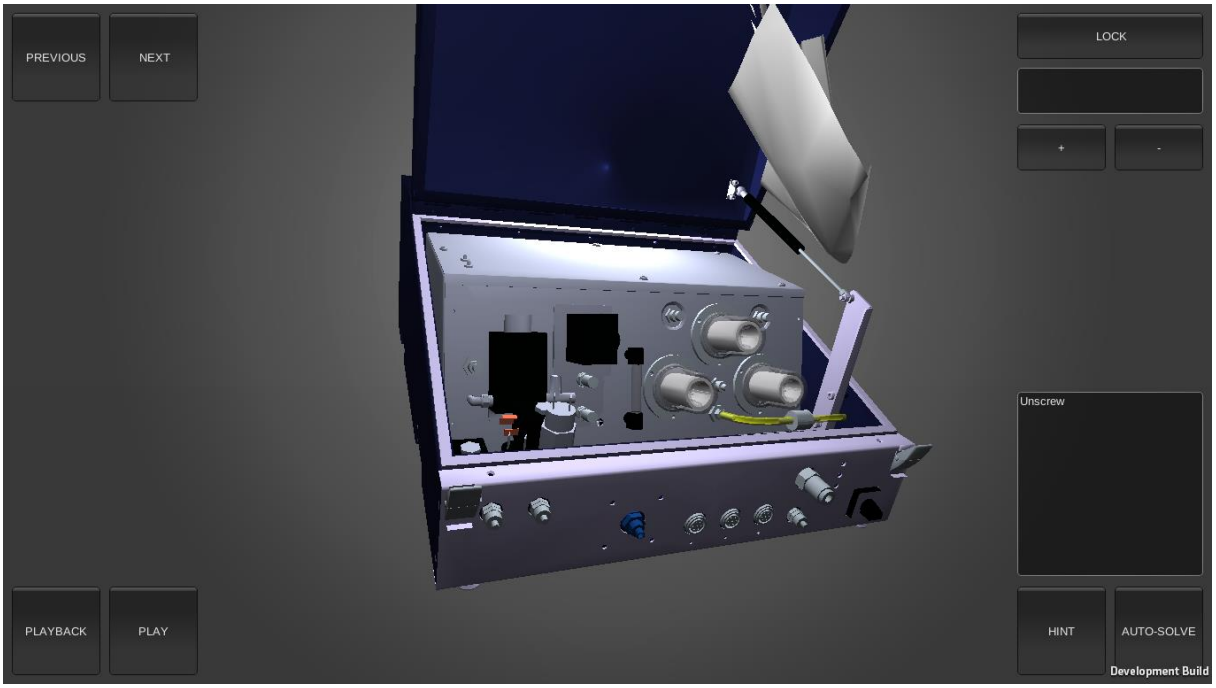
U gornjem lijevom kutu korisničkog sučelja nalaze se tipke za preskakivanje animacija trenutnog scenarija. Lijevom se bira prethodna a desnom sljedeća animacija.

U donjem lijevom kutu sučelja su tipke za kontrolu animacije. Lijevom tipkom animacija se izvodi unazad dok se desnom izvodi unaprijed. Obje tipke imaju mogućnost privremenog zaustavljanja animacije ako je ona u tijeku.

U donjem desnom kutu nalaze se također dvije tipke. Lijevom se uključuje i isključuje polje iznad tipke koje prikazuje smjernice za obavljanje trenutnog zadatka. Desna tipka pak aktivira sekvencu animacija koja prikazuje kronološki sve radnje koje radnik mora obaviti za trenutni scenarij.

Gornji desni kut sadrži polje u kojem piše ime trenutne označene komponente uređaja. Iznad tog polja stoji tipka kojom se omogućuju i onemogućuju rotacija modela i translacija kamere. To je naročito korisna opcija kada model zauzima veliki dio ekrana. Naime, tada može doći do istovremenog označavanja komponente modela i rotacije modela jer se unos obavlja u istom dijelu ekrana. Ispod polja s

trenutnom označenom komponentom nalaze se tipke za translaciju kamere. Lijevom se kamera približava modelu a desnom udaljava. Translacija kamere je animirana i kvantizirana.



Slika 3.4 Aplikacija za *Windows 8* platformu

4. Daljnja razmatranja

U budućem radu s Leap Motion uređajem preporuča se korištenje vlastitih implementacija detekcije gesta, uz što više predanalize dostupnih podataka za filtriranje lažnih i nepouzdanih podataka dostupnih s uređaja. Preporuča se anketiranje i ispitivanje potencijalnih korisnika ciljane aplikacije s prethodnih računalnim iskustvom o idealnim i intuitivnim gestama za određene akcije koje bi se pokretale na detekciju gesti Leap uređajem, kako bi se aplikacija osmislila na način da se u njoj radi jednostavno i intuitivno na prvi susret s korisnikom.

Za bolje rezultate, testiranje i nadahnuće moguće je isprobati i razmotriti tuđe javno dostupne kodove i aplikacije s istim uređajem.

Konkretno, u budućim projektima iste vrste kao što je ovaj, s istim potrebama kontrola i akcija, bilo bi dobro osmisliti druge geste, kako bi se akcije mogle proširiti, dodati nove funkcionalnosti bez preslikavanja na slične geste, tako da se ne bi slučajno izazvale akcije nepoželjne u tom trenutku.

Ostvarene geste i cijelu aplikaciju je potrebno testirati na većem broju korisnika prije javne objave postignuća, kako bi se osigurala njena jednostavnost, intuitivnost i efikasnost u izvršavanju zadanog cilja.

Kako bi se poboljšala upotreba glasovnih naredbi u web preglednicima, u daljnjem radu potrebno je proučiti korištenje dodatnih algoritama za uspoređivanje prepoznatih i ciljnih riječi, te testirati u različitim uvjetima: s više i manje šuma, te različitim kvalitetama mikrofona. Uz poboljšanja performansi, potrebno je proširiti funkcionalnosti s glasovnim naredbama koje bi bolje opisivale konkretne scenarije u grafičkom priručniku.

5. Zaključak

Osnovnom idejom korištenja kamera s dubinskom percepcijom za praćenje i najmanjih pokreta prsti *Leap Motion* ostvaruje impresivan koncept i pokazuje veliki potencijal za buduće aplikacije. Kada se izađe izvan demo aplikacija i stvarno počne raditi s konkretnim sučeljima, samo dio gesti je stvarno intuitivno, te postane očito tek nakon nekoliko sati 'igranja'. Iako postoje trenuci kada aplikacija i uređaj potpuno dobro rade zajedno, kada Leap uređaj ne 'odluči' da je ruka predaleko ili preblizu, te da u prostoru u kojem se uređaj koristi nije presvijetlo za izvršavanje akcija. S druge strane jednako često se događa da geste nisu dovoljno dobre (npr. prsti su preblizu a trebaju biti odvojeni, ruka je pod krivim kutom ili pak šaka ruke nije dovoljno zatvorena) i korisnik gubi vrijeme pokušavajući odraditi jednostavnu akciju kao npr. pritisak gumba. U takvim trenucima korisnici bi se mogli u početku obeshrabriti.

Dobro promišljena sučelja aplikacija s *Leap Motion* kontrolama (naredbama) će sigurno biti korak prema futurističkim aplikacijama, no trenutno većina aplikacija koristi 'trikove' (geste) koje nisu uvijek pouzdane za korištenje. Za sada Leap Motion je daleko od zamjene klasičnih ulaznih uređaja miša i tipkovnice, no sigurno ima svjetlu budućnost, posebno u slučajevima gdje olakšava korištenje same aplikacije. Iako vrlo interesantan koncept, ovo je tehnologija koja će zahtijevati još jedno vrijeme za daljnji razvoj i usavršavanje.

Pređemo li na glasovne naredbe u web preglednicima, dobijemo opet sličnu situaciju. Novi HTML5 *Web Speech API* je jako zanimljiv i ima vrlo koristan potencijal, no trenutno raspoznavanje govora nema dovoljno preciznu točnost i zahtijeva daljnje unapređenje APIa i/ili aplikacija uz korištenje dodatnih algoritama za uspoređivanje prepoznate riječi i moguće naredbe, no API je novi i daje obećavajuće rezultate za specifikaciju u bliskoj budućnosti.

6. Literatura

1. Glen Shires, *Voice Driven Web Apps: Introduction to the Web Speech API*, 14.1.2013., http://updates.html5rocks.com/2013/01/Voice-Driven-Web-Apps-Introduction-to-the-Web-Speech-API, 16.10.2013.
2. Brandon Satrom, *Using voice to drive the web: Introduction to the Web Speech API*, 14.1.2013., http://www.adobe.com/devnet/html5/articles/voice-to-drive-the-web-introduction-to-speech-api.html, 18.10.2013.
3. Ivan Lazarevic, *Voice command recognition using Levenshtein distance and Web Speech API*, 13.05.2013., http://blog.supplyframe.com/2013/05/13/voice-command-recognition-levenshtein-web-speech-api/, 13.12.2013.
4. Glen Shires, Hans Wennborg, Google Inc., *Web Speech API Specification*, 19.10.2012., https://dvcs.w3.org/hg/speech-api/raw-file/tip/speechapi.html, 16.10.2013.
5. Ricardo Cabello, *Three.js JavaScript library – Example: Loader Collada Keyframe*, http://threejs.org/examples/#webgl_loader_collada_keyframe, 2.12.2013.
6. Stemkoski *Three.js Examples*, 23.8.2013., http://stemkoski.github.io/Three.js/#leapmotion-visualization, 03.12.2013.
7. *Leap Motion Controller*, https://www.leapmotion.com/, 12.10.2013.
8. *Leapjs - Leap motion JavaScript library*, http://js.leapmotion.com/, 10.12.2013.
9. Lee Hutchinson, *Hands-on with the Leap Motion Controller*, http://arstechnica.com/gadgets/2013/07/hands-on-with-the-leap-motion-controller-cool-but-frustrating-as-hell/, 13.12.2013.
10. *Using microphone input in Unity3D*, http://www.kaappine.fi/tutorials/using-microphone-input-in-unity3d/, studeni 2013.
11. *Fundamental frequencies and detecting notes*, http://www.kaappine.fi/tutorials/fundamental-frequencies-and-detecting-notes/, studeni 2013.

12. *UDP Voice Recognition Server*, http://forum.unity3d.com/threads/172758-Windows-UDP-Voice-Recognition-Server, studeni 2013.
13. *List of speech recognition software,* http://en.wikipedia.org/wiki/List_of_speech_recognition_software , prosinac 2013.
14. *Sound pattern matching using Fast Fourier Transform in Windows Phone*, http://developer.nokia.com/Community/Wiki/Sound_pattern_matching_using_Fast_Fourier_Transform_in_Windows_Phone , prosinac 2013.
15. Specifikacije i mogućnosti uređaja Leap Motion, datum nastanka: 13.1.2013., https://www.leapmotion.com/product, datum pristupa: 15. 11. 2013.
16. Podatci o dostupnim strukturama podataka Leap Motion kontrolera za programski jezik C# i razvojnu okolinu Unity Engine, datum nastanka: 13.1.2013., https://developer.leapmotion.com/documentation/Languages/CSharpandUnity/Guides/Leap_Overview, datum pristupa: 15. 11. 2013.
17. Pristup bibliotekama za razvoj aplikacija korištenjem uređaja Leap Motion, datum nastanka: 13.1.2013., https://developer.leapmotion.com/downloads, datum pristupa: 15. 11. 2013.
18. Pristup projektima s Leap uređajem u Unity Engine-e, datum nastanka: 13.1.2013., https://developer.leapmotion.com/documentation/Common/Unity_Examples.html, datum pristupa: 15. 11. 2013.
19. OpenGL Programming Guide: Selection and Feedback, 13.2.2009., http://www.glprogramming.com/red/chapter13.html, 15.10.2013.
20. Coding a simple Octree, 5.1.2013., http://www.brandonpelfrey.com/blog/coding-a-simple-octree/, 27.10.2013.
21. Octree, 8.12.2013., http://en.wikipedia.org/wiki/Octree. 15.10.2013.
22. Čupić, M., Mihajlović, Ž., Interaktivna računalna grafika kroz primjere u OpenGL-u, 28.2.2013.

23. Group of authors, three.js – documentation, http://threejs.org/docs/, 15.10.2013
24. António Ramires Fernandes, Picking tutorial, http://www.lighthouse3d.com/opengl/picking/index.php?color1, 18.10.2013
25. Group of authors, OpenGL Programming/Object selection, 14.10.2013, http://en.wikibooks.org/wiki/OpenGL_Programming/Object_selection, 21.10.2013