Izrada 3D grafičkih elemenata za web stranicu pomoću biblioteke Three.js

Slukan, Tomislav

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

ELEKTROTEHNIČKI FAKULTET

Preddiplomski stručni studij Računarstvo

IZRADA 3D GRAFIČKIH ELEMENATA ZA WEB STRANICU POMOĆU BIBLIOTEKE THREE.JS

Diplomski rad

Tomislav Slukan

Osijek, 2023.

Sadržaj

1. Uvod

U ovom diplomskom radu govoriti će se o tome na koji način je moguće prikazati 3D modele na internetu u okviru web preglednika. Biti će pojašnjen način izrade samog 3D modela te primjene materijala i tekstura na isti. Osim pojašnjavanja dobre prakse i pravila modeliranja, u praktičnom dijelu rada moći će se vidjeti primjena same Three.js biblioteke te nove funkcionalnosti u usporedbi s današnjim web stranicama.

U drugom poglavlju govori se o područjima primjene 3D računalne grafike na internetu. Nakon toga važno je shvatiti osnovne koncepte o 3D modeliranju, stoga je bitno izdvojiti poveznicu između stvarnog i virtualnog svijeta, što je model, osnovna pravila modeliranja, što su teksture i materijali te osnovni pojmovi poput sjenčara, sjene te formata samih 3D modela. U četvrtom poglavlju govoriti će se o samoj Three.js biblioteci. Bit će odgovoreno na pitanja poput načina kreiranja projekta, što je scena i njezini glavni elementi poput svjetla, kamere, renderera, načina prikaza 3D modela te transformacije istih. U petom poglavlju bit će navedeno na koji se način pristupa rješavanju problema, web stranice za inspiraciju, odabir odgovarajućeg programa za 3D modeliranje te koji web preglednik je potrebno koristiti za testiranje projekta. U šestom poglavlju bit će riječ o prikazu rezultata u vidu koda i strukture projekta. U sedmom poglavlju bit će prikazan rezultat projekta u vidu same web stranice i prikazat će se sve njezine funkcionalnosti. U zadnjem, osmom poglavlju, bit će predstavljen problem koji se često javlja te njegovo rješenje za korištenje Three.js biblioteke. Na kraju je dan zaključak rada.

2. Pregled područja 3D računalne grafike na WEB-u

Three.js je JavaScript biblioteka[1] i sučelje za programiranje aplikacija (API[2]) koje služi za kreiranje i prikazivanje animirane 3D računalne grafike za većinu modernih web preglednika koje podržavaju WebGL i ubrzanje sklopovlja (eng. *hardware acceleration*). (tablica 1). Biblioteka ima otvoreni izvorni kod na GitHub servisu sa trenutno 82 500 zvijezdica i više od 32 000 forkova. Nastala je 2010. godine. [1], a razvio ju je Ricardo Cabello pod aliasom „Mr Doob“. [2] Službena dokumentacija nalazi se na web stranici *three.js.org/docs*. Korisnici često miješaju Three.js sa WebGL-om. Three.js biblioteka koristi WebGL kako bi prikazala 3D modele u web pregledniku.

Tablica 2.1 Popis preglednika koji podržavaju WebGL [3]

Naziv internet preglednika	Prva podržana verzija WebGL-a	Datum izdavanja
Google Chrome	8-32+	12.12.2010.
Firefox	4-23+	22.3.2011.
Opera	12.1-18+	5.11.2011.
Safari	5.1-7.1+	20.7.2011.
Internet Explorer	11+	17.10.2013.
Microsoft Edge	12-18+	29.7.2015.

WebGL je JavaScript API i ujedno preteča Three.js-a. Da bismo napravili bilo što korisno u WebGL-u potrebno je napisati jako puno koda. Tada stupa Three.js na scenu jer pomoću istoga na puno lakši i intuitivniji način možemo napraviti kompleksnije geometrije i scenu sa kamerama, izvorima

[1] Programska biblioteka ili knjižnica predstavlja pojam u programiranju koji označava zbirku pomoćnih modula koji nisu samostalni te nude rješenja tematski povezanim problemima.
Izvor: https://hr.wikipedia.org/wiki/Programska_knji%C5%BEnica (8 lipnja 2022., 12:13h)

[2] API (eng. „*Application Programming Interface*“) – skup pravila i specifikacija koje je potrebno slijediti kako bi bili u mogućnosti služiti se uslugama ili resursima operacijskog sustava ili nekog drugog izvora u smislu procedura, objekata, podataka i sl.
Izvor: https://hr.wikipedia.org/wiki/API (8 lipnja 2022., 12:01h)

svjetlosti, sjenama, materijalima, teksturama itd. Three.js biblioteka objedinjuje i pruža korisniku metode koje bi inače morao ručno pisati ukoliko bi želio koristiti WebGL direktno. [4]

Tvrtke koje koriste ovu biblioteku su Scale, Foretag, Teespring itd. Najčešće se koriste za prikaz personalizirane odjeće za web trgovine, interijera, unutarnjih dijelova kućišta za razne uređaje, animacije i sl. [5]

Popularna alternativa Three.js biblioteci koja se često koristi je još moderniji Babylon.js koji još više smanjuje WebGL apstrakciju te je puno brži. Iz tog razloga je pristupačniji za početnike.[6]

Postoji više biblioteka i okvira za prikaz 3D grafike u web preglednicima, ali se skoro svi baziraju na JavaScriptu ili WebGL pipelineu[3]. Moguće je na neke druge načine prikazati 3D modele u web preglednicima, ali su okviri i knjižnice bazirane na JavaScriptu najpopularniji i kvalitetniji za današnje standarde. Na primjer, koriste se WebAssembly ili „*Wasm*“ iz 2017. godine kojeg jezici poput C/C++ mogu *kompajlirati* za pokretanje na web pregledniku bez dodatnih dodataka ili ekstenzija. Dostupno je na Firefoxu, Google Chromeu, Safariju i Microsoft Edgeu. Za sada postoji samo 32-bitna verzija dok je 64-bitna u izradi.[7] S druge strane, ukoliko se žele izrađivati 3D igrice ili prikazivati 3D modeli, namijenjeni za navedene web preglednike, koristeći druge programske jezike poput .NET/C#, C++, Java, Python moramo znati kako to baš i nije najbolja ideja jer klijentska strana nativno radi samo pomoću JavaScripta ili HTML5, dok se za serversku stranu trebaju koristiti dodatci. Također, za klijentsku stranu ograničeni smo na Javi (*applet*[4]), Flashu, .NET (Silverlight), JavaScriptu te moramo znati da su neki od njih (poput Flasha ili Java *appleta*) izumrli ili se više ne podržavaju zbog sigurnosti. [8]

Web stranica koja mi se osobno najviše svidjela te ima potrebu za ovakvom tehnologijom je Centar znanja o raku dojke (izvor: https://interaktiv.brustkrebs.de/) iz Njemačke. Web stranica pruža 3D vizual vezano za prevenciju raka. Korisnik ima interakciju sa 3D modelom. Također, jedna od web stranica koju sam višestruko koristio je Sketchfab (izvor: https://sketchfab.com/), koja pruža 3D prikaz 3D modela različitih dizajnera širom svijeta.

---

[3] *Pipeline* predstavlja skup elemenata za obradu podataka u seriji pri čemu je izlaz jednog elementa ulaz sljedećeg. Često se ti elementi izvode paralelno ili mogu biti vremenski odvojeni.
Izvor: https://en.wikipedia.org/wiki/Pipeline_(computing) (8. lipnja 2022., 15:08h)

[4] *Applet* je svaka manja aplikacija koja obavlja određeni zadatak koji se izvodi u okviru *widgeta* ili većeg programa, kao dodatak.
Izvor: https://en.wikipedia.org/wiki/Applet (8.lipnja 2022., 15:38h)

Kako bi mogli vidjeti 3D modele u web pregledniku sa više slika u sekundi (eng. *Frames per second*) potrebno je u postavkama uključiti opciju ubrzanje sklopovlja (eng. *Hardware acceleration*). Današnja računala i web preglednici trebali bi, u velikoj većini, moći vrlo dobro prikazati 3D scenu u web pregledniku. Za Three.js nisam pronašao službene zahtjeve, ali sam pronašao za WebGL. Preporučuje se minimalno 2GB RAM-a, i grafička kartica koja podržava OpenGL 3.0 (shader model 4.0) te grafička kartica od 1GB RAM-a. Pošto je WebGL starija verzija biblioteke, mogu se primijeniti jače, ali ne zamjetne specifikacije na Three.js biblioteku. [9]

Naime, Three.js relativno se lako može naučiti, potrebno je savladati osnovne koncepte o programiranju i modeliranju. Ukoliko se zateknemo u situaciji da ne shvaćamo njihovu službenu dokumentaciju, na internetu postoji velika baza video uradaka i primjera kodova, gotovih projekata. I sam Ricardo Cabello snima korak po korak cijelu Three.js biblioteku u vidu tutorijala, ali se ova serija istih - plaća. Osim pregršt materijala na internetu nalazi se i dosta foruma gdje se mogu pronaći dragocjene informacije. Three.js ima dobre performanse, omogućava PBR render (eng. *Physically based rendering*), podržava popularne formate za uvoz 3D modela (.gltf, .glb, .fbx, .obj itd.), a sadrži i svoj uređivač (eng. *editor*) gdje se mogu uvesti svoji 3D modeli, kao i namještati različita svojstva scene. Važno je zapamtiti da Three.js biblioteka nije napravljena da bude *game engine*! [10] [11]

Po mom mišljenju Three.js će se sigurno više i kvalitetnije prikazivati na budućim web preglednicima jer će i računala postati jača, internet će postati još brži te će se moći više toga memorijski pohraniti u scenu. Iako i danas imamo čuda napravljena u ovoj tehnologiji smatram da nismo došli do samog vrha te ima mjesta za nadogradnju jer i sama 3D, VR i AR industrija cvjeta. Također moguće je raditi i ubaciti razne animacije u same modele te imati interakciju s korisnicima. Osim prikaza 3D modela i animacije imamo podršku za VR (virtualna stvarnost, eng. *Virtual Reality*) i AR (proširenu stvarnost, eng. *Augmented Reality*) tehnologiju koju je moguće ostvariti u web pregledniku pomoću Three.jsa i WebXR API-ja.

3. Općenito o 3D modeliranju

Tema ovog rada je objasniti osnovne principe 3D modeliranja koje plijeni sve više pozornosti u svijetu. Ima široku primjenu u industriji na gotovo svim poljima. Kako se u ovom radu najviše pozornosti posvećuje razvoju web stranica, ali i 3D modeliranju, spojilo se sve navedeno u jedno te se pronašao način da se prezentiraju 3D modeli na internetskoj stranici. Kako bi ovaj rad imao i svoju praktičnu primjenu, na portalu Stup, koji služi studentima kako bi pronašli praksu, prezentirat će se najmanje 3 modela pomoću biblioteke Three.js. Kako se spomenuti portal u sklopu fakulteta redizajnira navedeno će mu doći kao pravo osvježenje jer je Three.js budućnost modernih i lijepih internetskih web stranica. Modeli će se modelirati u programu 3ds Max, koji je besplatan za sve studente (moguće je aktivirati studentsku licencu ukoliko studirate na bilo kojem sveučilištu u svijetu). Za primjenu modela koristit će se biblioteka Three.js koja je otvorenog koda (besplatna). 3ds Max je odabran zbog raznovrsne primjene. Također, Three.js je biblioteka koja je relativno nova, ali uvelike olakšava posao koji je WebGL započeo mnogo vremena ranije.

3.1. Poveznica između stvarnog i virtualnog svijeta

Ljudi percipiraju svijet oko sebe u tri dimenzije - širina, dužina i visina. Kako je prvobitna grafika na računalima bila isključivo 2D, a zbog sve većeg razvoja tehnologije i ljudskih potreba javila se potreba za uvođenjem 3D grafike, modeliranjem, printanjem i slično. Danas se nalazimo u svijetu gdje gotovo sve možemo simulirati na računalima prije nego napravimo eksperiment u stvarnosti, a na tome možemo zahvaliti između ostaloga i 3D grafici. 3D grafika ima vrlo široku primjenu u gotovo svim industrijama te nam uvelike olakšava posao i predodžbu problema. Svaku stvar iz realnog svijeta možemo rekreirati u nekom 3D programu po želji i to na više načina (npr. modeliranje, fotogrametrija). Na taj način se doslovno ideja iz ljudskog mozga može rekreirati u stvarnosti - u fizičkom obliku. Navedeno se primjenjuje se u virtualnoj stvarnosti, računalnim igrama, simulacijama, animacijama (filmska industrija, vizualni efekti), IT-u, medicini, građevini, raznim istraživanjima itd.

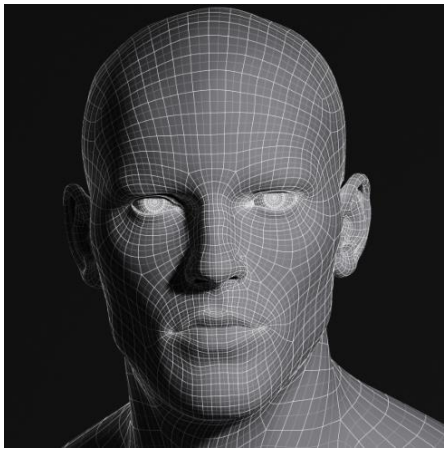
3.2. Što je 3D model?

3D model je matematička reprezentacije nečega trodimenzionalnog. Taj model se najčešće opisuje poligonima koji se sastoje od tri ili više točaka te zapisujemo njihov položaj u prostoru. U sljedećoj rečenici najjednostavnije će se objasniti kako funkcionira 3D modeliranje. Dakle, ukoliko se zamisli

neki objekt, koji želimo realizirati u 3D programu, i preko njega se „stavi“ mrežu, koja se sastoji od ravnomjernih kvadratića, upravo navedeno rezultirati će 3D modelom. Važno je da je ta mreža kvadratića (poligona) ravnomjerno raspoređena zbog raznovrsne primjene tog istog 3D modela i daljnjih postupaka u cijelom procesu. Sve navedeno može se staviti pod zajednički nazivnik, a to je topologija. Na primjer, ako želimo animirati 3D model, isti je potrebno definirati kosturom te ako mreža poligona nije ravnomjerno raspoređena po određenim pravilima animacija neće biti očekivana te se mogu se javiti razni artefakti i problemi u kasnijem korištenju samog modela. [12]

3.3. Pravila 3D modeliranja

Kada se želi napraviti 3D model vrlo je važno razmišljati nekoliko koraka unaprijed kako bismo si uvelike olakšali posao. U ovom odlomku napisati će se napisati zlatna pravila 3D modeliranja koja su vrlo važna i svi bi ih trebali koristiti. Vrlo je važno razlikovati alate koje ćemo koristiti za pojedini posao. Danas postoji jako puno programa za različite primjene u 3D modeliranju. O tome ovisi brzina i kvaliteta modela. Vrlo je važno fokusirati se na širu sliku. Svaki model može se napraviti od primitivnih (jednostavnih) oblika koje nadograđujemo u fazama. Potrebno je koristiti reference iz stvarnog svijeta, ukoliko je to moguće, kako bi imali dobre proporcije. Vrlo je važno ostaviti prostora za adaptacije, tj. moramo biti svjesni da ćemo neke dijelove modela u budućnosti morati promijeniti. Kako bismo bili što efikasniji dobro je spremati i ponovno iskorištavati već napravljene dijelove modela. Vrlo je važno fokusirati se na kvalitetu površine modela zbog svjetla. Zbog toga je važno imati dobar raspored poligona (topologije modela). Na kraju, kada se završi sa samim modeliranjem vrlo je važno imati, osim oka za detalje, prilikom samog modeliranja, oko za detalje u smislu tekstura ili materijala koje ćemo koristiti na modelu. [13]



Slika 3.1. Prikaz ravnomjerne mreže poligona koji opisuju 3D model [14]

3.4. Osnovni geometrijski oblici

Već je poznato kako svi ljudi razmišljaju na različite načine, a tako je i prilikom 3D modeliranja. Pošto su mogućnosti beskonačne, svatko na različit način može doći do sličnog (traženog) rješenja. Programi za 3D modeliranje nude korisnicima ne zahtjevne i jednostavne 3D objekte koji se nazivaju primitivi. Njihovim kombinacijama mogu se stvoriti nešto složeniji oblici te na taj način može se modelirati ono što se zamislilo. U kombinaciji sa primitivima uvelike nam pomažu metode poput zbrajanja, oduzimanja, množenja i sl. kako bismo dobili složeniji model. Primitive koji postoje u većini programa za 3D modeliranje su kocka, cilindar, sfera, stožac, torus. [15]

3.5. Razlika između low i high poly modela

Razlika između navedenih modela je u broju poligona koji čine sami model. *Low poly* model ima manji broj poligona nego *high poly* model. Naravno, postoje razlike kada se koriste jedne, a kada druge te zašto ih uopće koristimo. Kada se želi prezentirati 3D model na internetu ili u igricama vrlo je važna kompresija modela. Zadaća je da imamo što bolji model sa što više detalja, a da pri tome isti ne zauzima previše resursa računala. Taj problem se riješava na način da koristimo model sa drastično manje poligona gdje isti koristi model sa više poligona kao svoju masku. Na taj način štedimo resurse i taj model može se koristiti u stvarnom vremenu. S druge strane, modele sa više poligona koristimo kada, na primjer, radimo render (detaljnu fotografiju) nekog proizvoda iz blizine i pri tome ga ne koristimo u igrici, internetu i slično zbog uštede performansi. [16]

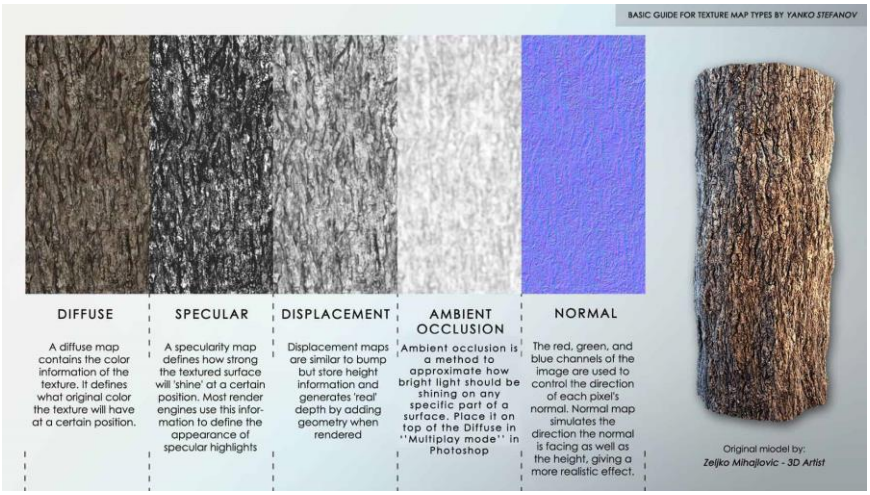
3.6. Teksture

Kada se završi 3D modeliranje dobiveni model predstavlja samo oblik – bez površinskih obilježja. Kako bi mu se dala obilježja stvarnog svijeta, poput boje ili uzorka potrebno je staviti teksture na isti. Tekstura je ništa drugo nego bitmapa, najčešće u .jpg ili .png formatu koja 3D modelu daje obilježja tj. izgled iz stvarnog svijeta. Tu bitmapu može se napraviti u nekom programu (najčešće se koristi *Photoshop*) ili fotografirati iz stvarnog svijeta. Na taj način model ima uzorke i boju te postaje realniji i zanimljiviji.

Slika 3.2. Tekstura; fotografija koja je dobra, ali nije realna kao materijal. [17]

3.7. Materijali

Za razliku od tekstura, materijalima se određuju optička svojstva modela. Na primjer, određuje se hoće li model biti sjajan, metalan, određene boje, na koji način će se svjetlost odbijati od njega i sl. Materijali su puno bolji izgledom od tekstura, ali i zahtjevniji za izradu. Materijali koriste više različitih bitmapa u isto vrijeme te su zbog toga realniji. Svaka bitmapa daje drugo svojstvo predmetu (metalni odsjaj, boju, dubinu i sl.). [18]



Slika 3.3. Materijal; sadrži više bitmapa s različitim obilježjima te zajedno čine realistični materijal. [19]

3.8. UV mapping & unwrapp

Kada imamo gotov 3D model na njega je potrebno primijeniti teksturu ili materijal kako bi model bio realističniji te služio svojoj svrsi. Postupak kojim „odmotamo“ poligone modela naziva se UV *unwrapping*, a postupak projekcije 2D teksture ili materijala na 3D model naziva se UV *mapping*. Postupak je vrlo jednostavan za shvatiti. Model se predstavlja pomoću x, y i z koordinata u prostoru. Kako bi se mogla primijeniti tekstura ili materijal na 3D model potrebno je 3D model predstaviti ravnom mrežom poligona (ukratko, izravnati 3D model na koji će se primijeniti teksturu ili materijal). UV predstavljaju horizontalnu i vertikalnu os 2D prostora kao x, y i z koordinate u 3D prostoru. Postupak odmotavanja sastoji se od toga da se na 3D modelu odabiru šavove (eng. *seems*) i to je mjesto koje se odvaja (reže) kako bi ga mogli izravnati. Kako na tim mjestima počinje tekstura ili materijal, njih se stavlja na manje vidljiva mjesta na modelu. Prije primjene same teksture ili materijala na već odmotan plašt 3D modela radi se provjera sa gotovim uzorcima - najčešće crno-bijelih kockica (eng. *checkers pattern*), a sve to kako se ne bi pojavile distorzije. [20]

3.9. Sjenčar

Sjenčar (eng. *shader*) predstavlja računalni program koji se obično izvodi na grafičkoj kartici računala. Njegova zadaća je da se izračunaju odgovarajuće razine svjetla i boje tijekom renderiranja 3D scene za svaki piksel. Sjenčari govore grafičkoj kartici kako se svjetlo treba ponašati na površini 3D modela, na koji način imaju interakcije sa materijalom, koje boje trebaju biti te ostale detalje i općenito računalnoj grafici daju realan izgled. Postoji više tipova sjenčara, a to su sjenčari piksela, točaka, geometrije, teselacijski, primitivni te sjenčari za praćenje zraka (eng. *Ray tracing*). [21]

3.10. Sjene

U računalnoj grafici objekti se često prikazuju bez sjene ukoliko su performanse scene ograničene ili sjene nisu potrebne za razumijevanje. Sjene prenose velike količine informacija jer predstavljaju drugi pogled na objekt. Točka je u sjeni, u odnosu na izvor svjetlosti, ako zrake iz tog istog izvora svjetlosti ne mogu izravno doći do točke. Kako bi se dodala sjena u 3D računalnoj grafici često se koristi mapiranje sjena ili projekcija sjena. Sjene se stvaraju na način da se testira je li piksel vidljiv iz izvora svjetlosti pomoću usporedbe piksela sa *z-bufferom* (poznatije kao *depth buffer*). Ovaj međuspremnik dubine koristi se za predstavljanje informacije o dubini objekta u 3D prostoru iz određene perspektive. Ovi međuspremnici predstavljaju pomoć pri renderiranju scene kako bi se osiguralo da ispravni poligoni ispravno zatvaraju druge poligone. [22]

3.11. Baking

Pečenje (eng. *baking*) se odnosi na proces spremanja informacija vezane za 3D model u 2D teksturu (bitmapu). Također, ovaj proces češće se koristi kada postoje dva 3D modela te se informacije prvog modela transformiraju na UV mapu drugog modela te se spremaju u teksturu. Svrha ovog procesa je izračunavanje informacija vezanih za teksture i svjetlo unaprijed. Ovaj postupak koristi se kada 3D modele izvozimo u *game engine*. Podaci koji se dobijemu pečenjem su ambijentalne sjene, informacije vezane za normale (detalji površine pohranjene kao smjerovi vektora), smjer (gore, dolje, lijevo, desno itd.), zakrivljenost (rubovi, šupljine), položaj geometrije i sl. Jedna velika prednost postupka pečenja je da možemo sačuvati sve detalje *high poly* verzije modela, a u *game engineu* možemo tu istu mapu staviti preko *low poly* verzije modela. Na taj način štedi se memorija i snižavaju performanse, a da pri tome postoje svi potrebni detalji modela.[23]

3.12. Formati

3D formati koriste se za spremanje informacija o modelu i predstavljanje istih.

.glTF (eng. *Graphics Language Transmission Format*) predstavlja datoteku koja sprema informacije o 3D modelu u JSON formatu. JSON (eng. *JavaScript Object Notation*) je standardni format baziran na tekstualnom zapisu za prezentiranje strukturiranih podataka u JavaScript sintaksi za prenošenje podataka u web aplikacijama. JSON smanjuje veličinu 3D modela te vrijeme potrebno za otvaranje i korištenje istog. Razlika između glTF i GLB formata je ta da GLB predstavlja binarni spremnik glTF-a. Ukratko, GLB povezuje sve teksture i podatke o modelu u jednu datoteku, dok glTF sadrži više odvojenih datoteka (za teksture i model). [24]

3.12.1. OBJ

.OBJ format je jednostavan format podataka koji sadrži informacije samo o 3D modelu. Ovaj format bilježi položaje svake točke modela, UV položaje koordinata teksture, normale koje čine svaki poligon modela te informacije za svako lice koje čini mrežu poligona (listu točaka) 3D modela. Ovaj format sadrži animacije i informacije vezane uz svjetlo ili poziciju scene. [25]

3.12.2. FBX

.FBX (eng. ***F**ilm**b**o**x***) je format kojeg posjeduje Autodesk od 2006. godine. Koristi se za razmjenu 3D geometrije i animacije. Ovaj format koristi se u filmovima, igricama, proširenoj i virtualnoj stvarnost. [26]

4. Biblioteka Three.js

4.1. Načini pokretanja projekta

Postoji više načina instaliranja Three.js biblioteke u naš projekt. U ovom radu opisat će se tri metode koje su najkorištenije.

4.2. Pomoću URL-a

Kako bismo uključili biblioteku Three.js u naš projekt pomoću URL-a (hiperveze) potrebno je stvoriti osnovnu strukturu projekta u HTML-u te unutar <body> tagova napisati slijedeću liniju koda:

```
<script src="https://threejs.org/build/three.js"></script>
```

Slika 4.1. Pokretanje biblioteke pomoću URL-a.

Nakon toga potrebno je otvoriti web stranicu pomoću Live servera (ekstenzija u *Visual Studio Codu*) ili instalirati *Xampp* (program koji omogućava simulaciju web servera na računalu).

4.3. Lokalna kopija Three.js

Na sličan može se uključiti Three.js biblioteka samo što ona neće biti dohvaćena preko URL-a nego će se lokalno pohraniti u „three.js“ datoteku te ju pozvati unutar <script> taga, slično kao u prvom primjeru. Postupak je da se u projektu stvori novi direktorij proizvoljnog imena (npr. „js“) te se unutar njega napravi „three.js“ JavaScript datoteka. Unutar te datoteke će se sadržaj s linka kojeg se koristilo u prvom primjeru kopirati (CTRL+C) te zalijepiti u „three.js“ datoteku (CTRL+V). Nakon toga će se pomoću sljedeće linije koda uključiti „three.js“ datoteka u „index.html“. Na ovaj način dobije se lokalno spremljena skripta na računalu u projektu. Sadržaj datoteke „three.js“ može se pronaći na sljedećem linku službene dokumentacije u sekciji „**code“** > „**download“**: https://threejs.org/

```
<script src="js/three.js"></script>
```

Slika 4.2. Pokretanje biblioteke pomoću lokalne kopije

4.4. Instalacija pomoću NPM-a

Kako bismo instalirali Three.js biblioteku pomoću NPM-a (*Node Packet Manager* – koristi se za upravljanje ovisnostima na serverskoj strani, upravljanje paketima i verzijama. Dodatno, dodaje strukturu u projekt) potrebno je otvoriti terminal u programu po izboru u kojem se piše kod, u ovom

slučaju to je već ranije spomenuti *Visual Studio Code*. Unutar terminala piše se naredba **npm init**. Nakon odrađenih koraka može se vidjeti da se u direktoriju stvorila datoteka „*package.json*“. Kako bismo preuzeli biblioteku Three.js potrebno je napisati naredbu: **npm install three**. Nakon toga uspješno se preuzela biblioteka Three.js te ju je potrebno uključiti u kod u „*index.html*“ datoteku. [27]

Prvi način je da se uključi cijela three.js biblioteka, gdje zvjezdica (*) predstavlja „*all*“ tj. sve:

```
import * as THREE from 'three';

const scene = new THREE.Scene();
```

Slika 4.3. Kako uključiti cjelokupnu biblioteku.

Drugi način je da se uključe samo oni dijelovi biblioteke koje su potrebni za rad, u ovom primjeru uključen je samo dio potreban za inicijalizaciju scene:

```
import { Scene } from 'three';

const scene = new Scene();
```

Slika 4.4. Kako uključiti pojedine dijelove biblioteke.

4.5. Scena

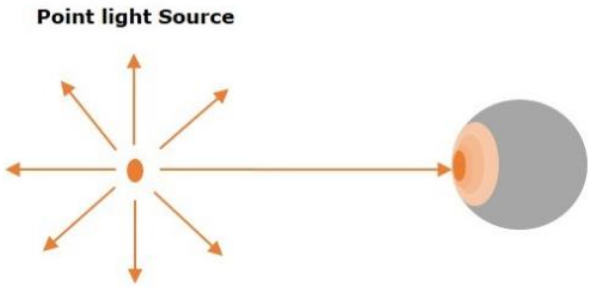
Kako bi se prikazalo bilo što pomoću Three.js biblioteke potrebno je stvoriti scenu na kojoj će biti prikazani objekti. Osim samih objekata u scenu je potrebno dodati kameru preko koje korisnik može vidjeti objekte na sceni, izvor svjetlosti te renderer koji služi za prikazivanje cjelokupne scene. [28]

4.6. Izvori svjetlosti

Biblioteka Three.js nudi na izbor nekoliko tipova svjetlosti koji se mogu koristiti u našem projektu, a koji su svojstvima temeljeni na stvarnom životu. Slično kao i u programima za 3D modeliranje (Blender, 3ds Max i sl.) postoje ambijentalno svjetlo, usmjereno svjetlo, svjetlo hemisfere, svjetlosna sonda, točkasto svjetlo, pravokutno svjetlo itd. U ovom radu objasniti će se izvori svjetlosti koje je autor koristio kao i one najčešće korištene. Za svaki projekt potrebno je testirati i isprobavati izvore svjetlosti kako bi se dobili željeni rezultati.

4.6.1. Točkasti izvor svjetlosti

Točkasto svjetlo (eng. *point light*) predstavlja izvor svjetlosti koje se emitira iz jedne točke u svim smjerovima. Primjer iz stvarnog svijeta bio bi izvor svjetlosti koji se dobije od žarulje. Navedeni izvor svjetlosti može projicirati sjenu. [29]



Slika 4.5. Primjer točkastog izvora svjetlosti [30]

```
const light = new THREE.PointLight( 0xff0000, 1, 100 );
light.position.set( 50, 50, 50 );
scene.add( light );
```

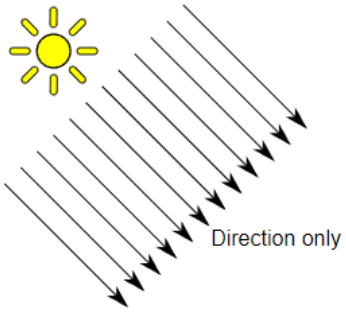
Slika 4.6. Primjer koda.

Objašnjenje koda:

PointLight(boja: Integer, intenzitet: Float, udaljenost: Number, gušenje: Float);

4.6.2. Usmjereni izvor svjetlosti

Usmjereni izvor svjetlosti (eng. *directional light*) je svjetlo koje emitira zrake u određenom smjeru. Ponaša se kao da je beskrajno daleko te da su zrake koje izlaze iz njega paralelne. Primjer je simulacija dnevnog svjetla, to jest Sunce. Također projicira sjene. [31]



Slika 4.7. Primjer usmjerenog izvora svjetlosti [32]

```
const directionalLight = new THREE.DirectionalLight( 0xffffff,
0.5 );
scene.add( directionalLight );
```

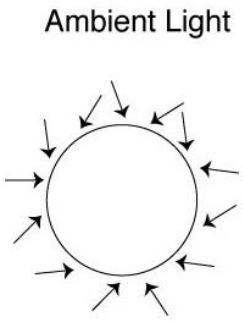
Slika 4.8. Primjer koda.

Objašnjenje koda:

DirectionalLight(boja: Integer, intenzitet: Float);

4.6.3. Ambijentalni izvor svjetlosti

Ambijentalni izvor svjetlosti (eng. *ambient light*) je izvor koji jednako osvjetljava sve objekte koji se nalaze u sceni. Ovaj izvor svjetlosti ne može se koristiti za projiciranje sjena jer nema smjer. [33]



Slika 4.9. Primjer ambijentalnog izvora svjetlosti [34]

```
const light = new THREE.AmbientLight( 0x404040 ); // soft white
light
scene.add( light );
```

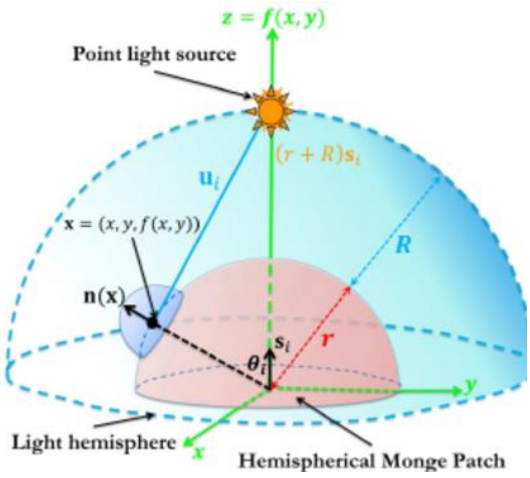
Slika 4.10. Primjer koda.

Objašnjenje koda:

AmbientLight(boja: Integer, intenzitet: Float);

4.6.4. Svjetlost hemisfere

Izvor svjetlosti hemisfere (eng. *hemisphere light*) je izvor svjetlosti postavljen izravno iznad scene, s degradiranjem boje - od boje neba do boje tla. Ne može se koristiti za projiciranje sjena. [35]



Slika 4.11. Primjer izvora svjetlosti hemisfere [36]

```
const light = new THREE.HemisphereLight( 0xffffbb, 0x080820, 1 );
scene.add( light );
```

Slika 4.12. Primjer koda.

Objašnjenje koda:

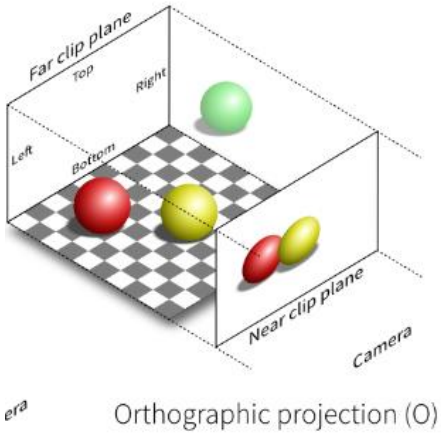
HemisphereLight(boja neba: Integer, boja tla: Integer, intenzitet: Float);

4.7. Kamere

Pomoću kamere korisnik može vidjeti objekte na sceni. Postoji nekoliko tipova kamera u Three.js biblioteci, a to su: kockasta, ortografska, perspektivna i stereo kamera. U nastavku će se objasniti dvije najkorištenije: ortografska i perspektivna kamera. Osim jedne kamere, može se postaviti skup kamera (eng. *array camera*) koje služe za učinkovito renderiranje scene s unaprijed definiranim skupom kamera. Navedeno predstavlja vrlo važan aspekt za renderiranje VR scena.

4.7.1. Ortografska kamera

U ovom načinu projekcije, veličina objekta nakon renderiranja ostaje konstantna bez obzira na njegovu udaljenost od kamere. Ovaj tip kamere može biti koristan za render 2D scena ili prikaz korisničkog sučelja. [37]



Slika 4.13. Primjer prikaza ortografske kamere. [38]

```
const camera = new THREE.OrthographicCamera( width / - 2, width /
2, height / 2, height / - 2, 1, 1000 );
scene.add( camera );
```

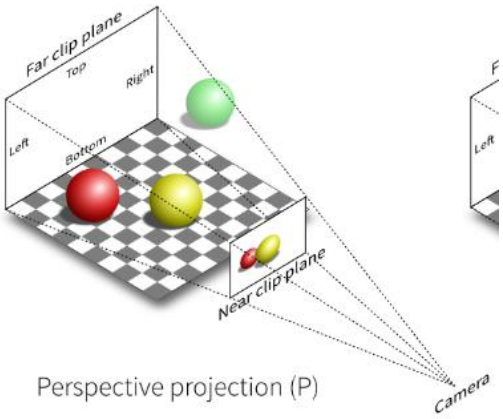
Slika 4.14. Primjer koda.

Objašnjenje koda:

OrtographicCamera(lijevo: Number, Desno: Number, gore: Number, dolje: Number, blizu: Number, udaljeno: Number);

4.7.2. Perspektivna kamera

Kamera koja koristi perspektivnu projekciju (eng. *perspective camera*). Ova kamera osmišljena je da oponaša ljudsko oko. Ovo je najčešći način projekcije koji se koristi za 3D renderiranje scene jer je prirodno. [39]



Slika 4.15. Primjer prikaza perspektivne kamere. [40]

```
const camera = new THREE.PerspectiveCamera( 45, width / height,
1, 1000 );
scene.add( camera );
```

Slika 4.16. Primjer koda.

Objašnjenje koda:

PerspectiveCamera(vidno polje: Number, aspekt: Number, blizu: Number, daleko: Number);

4.8. Učitavanje 3D modela u scenu

Prije svega potrebno je odabrati pravi format 3D modela kojeg se želi prezentirati na web stranici. Na internetu može se pronaći bezbroj modela koji su napravljeni u različitim formatima, ali se mora znati kako svaki format ima svoje prednosti i nedostatke. Three.js dokumentacija nalaže da je najbolje koristiti 3D modele koji imaju .GLTF ili .GLB ekstenziju zbog odlične podrške te fokusa na brzinu učitavanja modela. Ipak, ako su modeli u nekom drugom formatu (npr. .OBJ ili .FBX) u tom slučaju Three.js nudi posebne *loadere* pomoću kojih se i takvi modeli mogu koristiti u sceni jer se regularno ažuriraju i dostupni su u biblioteci Three.js, ali nisu preporučljivi.

Obavezno je korištenje lokalnog poslužitelja jer se mnoge uobičajene pogreške događaju prilikom učitavanja modela neispravnim hostingom datoteka.

Što se tiče zadanih *loadera* koji se dobijaju u Three.js biblioteci, nudi se *ObjectLoader*, a ostale je potrebno dodavati individualno u kod.

```
import { GLTFLoader } from 'three/addons/loaders/GLTFLoader.js';
```

Slika 4.17. Primjer .GLTF *loadera* koji nije zadani, te ga je potrebno individualno dodati u kod.

Nakon dodavanja *loadera* u kod, mogu se dodati modeli u scenu. Svaki *loader* ima drugačiju sintaksu, ali su promjene minimalne i logične, s time da postoje tutorijali ukoliko je potrebna pomoć s istima. [41]

```
const loader = new GLTFLoader();

loader.load( 'path/to/model.glb', function ( gltf ) {

    scene.add( gltf.scene );

}, undefined, function ( error ) {

    console.error( error );

} );
```

Slika 4.18. Primjer sintakse prilikom učitavanje za .GLTF format modela.

4.9. Transformacije

Biblioteka Three.js koristi matrice za dekodiranje transformacija 3D modela. Transformacije 3D modela podrazumijevaju promjenu položaja, rotaciju i skaliranje objekta na sceni. Svaka instanca objekta sadrži matricu koja sprema i ažurira ove vrijednosti. Postoje dva načina na koje se mogu promijeniti transformacije objekta, a to je da se izmijene navedene vrijednosti za pojedini objekt te se puste biblioteci da ponovno izračuna matricu objekta, a druga metoda je da se izmjene svojstva matrice objekta izravno.

U 3D računalnoj grafici najčešće se upotrebljava 4x4 matrica kao matrica transformacije koje se koriste u WebGL-u. Navedeno omogućava Vector3 koji omogućuje točki u 3D prostoru da podlegne transformacijama poput translacije, rotacije, smicanja, skaliranja, refleksije i sl. tako što se navedeno množi matricom. Izraz je također poznat kao primjena matrice s vektorom. [42]

4.10. WebGL renderer

Zadaća WebGL renderera je prikazati scenu pomoću WebGL-a. Konstruktor izgleda ovako: WebGLRenderer (parametri: Objekt), gdje sami parametri nisu obavezni. Ukoliko ih nema, renderer će pretpostaviti neke zadane vrijednosti. Vrijednosti koje se mogu unositi kao parametri su platno (eng. *canvas*); mjesto gdje rederer prikazuje samu scenu. Ako se ne unese kao parametar, stvorit će se novi element platna. Osim platna postoje: kontekst, preciznost sjenčara (highp, mediump, lowp), alfa (kontrolira čistu alfa vrijednost, zadana vrijednost je nula), dubina, matrica (eng. *stencil*), dubina i sl. [43]

4.11. DRACOLoader

Draco je biblioteka koja je otvorenog koda te se koristi za kompresiju i dekompresiju 3D modela. Ono što DRACO nudi je znatno manja geometrija, ali po cijeni dodatnog vremena dekodiranja na klijentskoj strani. Navedene datoteke ne sadrže informacije o boji, teksturi, animaciji i sl. Normalna .GLTF datoteka može se pretvoriti u komprimiranu .GLTF datoteku pomoću .GLTF *pipelinea.* [44]

5. Analiza problema

5.1. Kako sam pristupio rješavanju problema

Ovom problemskom zadatku, s praktičnim djelom, pristupilo se na način da se isti rastavio na nekoliko manjih dijelova.

U prvom dijelu rada modelirali su se 3D modeli u programima 3ds Max i Blender. Blender se koristio samo za jedan model, a to je bio model studenta jer se prilikom modeliranja koristio „*sculpting*“. To je način modeliranja gdje se ne manipulira točkama, rubovima ili površinom poligona nego se koriste razni kistovi te na taj način lakše i brže modeliraju prirodni dijelovi modela. Svi ostali modeli modelirali su se u 3ds Maxu. Nakon završenog procesa modeliranja na modele stavili su se materijali ili teksture, ovisno o želji i krajnjem rezultatu. Nakon toga model studenta animirao se u željeni oblik.

Sve te datoteke koje čine prezentacijsku web stranicu stavilo se na besplatni web hosting (Netlify). Nakon svake promjene koda ili pisanog dijela završnog rada, datoteke su se podigle na javni GitLab repozitorij. Za pisanje koda koristio se besplatni *Visual Studio Code* s ekstenzijom „*Prettier*“ koja omogućuje brzo formatiranje koda kako bi sam kod bio uredno i lijepo napisan.

5.2. Primjeri web stranica dizajniranih pomoću biblioteke Three.js

Three.js je JavaScript biblioteka koja je uvelike olakšala prikazivanje 3D modela u modernim web preglednicima. Sve što je potrebno je napraviti ili kupiti 3D model i u nekoliko linija koda inicijalizirati scenu zajedno s tim modelom. Kako je vrlo jednostavno i daje moderan izgled web stranicama, smatra se da je to budućnost web razvoja. Postoji dosta lijepih i modernih web stranica sa nizom drugačijih mogućnosti koje mijenjaju korisničko iskustvo na bolje (razmišljanje izvan kutije). Na internetu se najviše pojavljuju portfolio web stranice koje koriste ovu tehnologiju, igre poput šaha i sl. te web stranice koje se koriste u edukacijske svrhe. U nastavku će se istaknuti imena i poveznice do primjera web stranica koje su se autoru ovog rada najviše svidjele.

- Chessboard3.js, veza na primjer: https://jtiscione.github.io/chessboard3js/index.html
- Već spomenut Centar znanja o raku dojke iz Njemačke, veza na primjer: https://interaktiv.brustkrebs.de/
- Portfolio Bruno Simon, veza na primjer: https://bruno-simon.com/

Iz ovih primjera može se vidjeti raznovrsnost biblioteke Three.js te da je granica samo mašta programera.

5.3. Odabir programa za 3D modeliranje

Od svih programa koji se koriste za 3D modeliranje (između ostalog) odlučilo se na korištenje 3ds Maxa i Blendera. 3ds Max je vrlo svestran program koji se koristi u industriji duži niz godina. On nije otvorenog koda poput programa Blender, ali je besplatan za sve studente diljem svijeta uz predočenje potvrde studiranja na njihovu mail adresu. Uz 3ds Max (Autodesk) poznati su još Maya (također Autodesk) i Houdini (SideFX). Svi navedeni programi omogućuju 3D modeliranje, animaciju, renderiranje, vizualne efekte, primjene u virtualnoj i proširenoj stvarnosti, filmske efekte i još mnogo toga. Također, imaju puno ekstenzija trećih strana koje olakšavaju posao na profesionalnoj razini. Uz 3ds Max često se koristi renderer „*Arnold*“ koji se koristi za render fotorealističnih scena. 3ds Max se odabrao pri pisanju ovog rada, jer je to bio program koji je autor ovog rada koristio prilikom savladavanja 3D modeliranja, te su istom sve stvari uvelike bile poznate. Blender se koristio samo za jedan model studenta i to zbog besplatnog („*sculpting*“) načina rada (pomoću kistova) kojeg 3ds Max nema, nego bih se trebao koristiti dodatni program, naziva „*Zbrush*“, koji nije besplatan i koji je zahtjevniji od Blendera.

5.4. Odabir web preglednika

Google Chrome je izabran kao web preglednik iz razloga što njega, autor rada, koristi u svim situacijama, također, autor rada navikao je na isti, te smatra kako je upravo taj preglednik moderan. Ono što je najvažnije je to da Three.js radi na većini današnjih web preglednika koji podržavaju hardversko ubrzanje (grafičko renderiranje) koje transferira svu grafiku i tekst sa procesora na grafičku procesorsku jedinicu u računalu. Uvjet je da web preglednik podržava WebGL. Osim Google Chromea može se koristiti većina najkorištenijih web preglednika današnjice poput Firefoxa, Opere, Safarija, Internet Explorera 11 te Microsoft Edgea (pogledati tablicu 1).

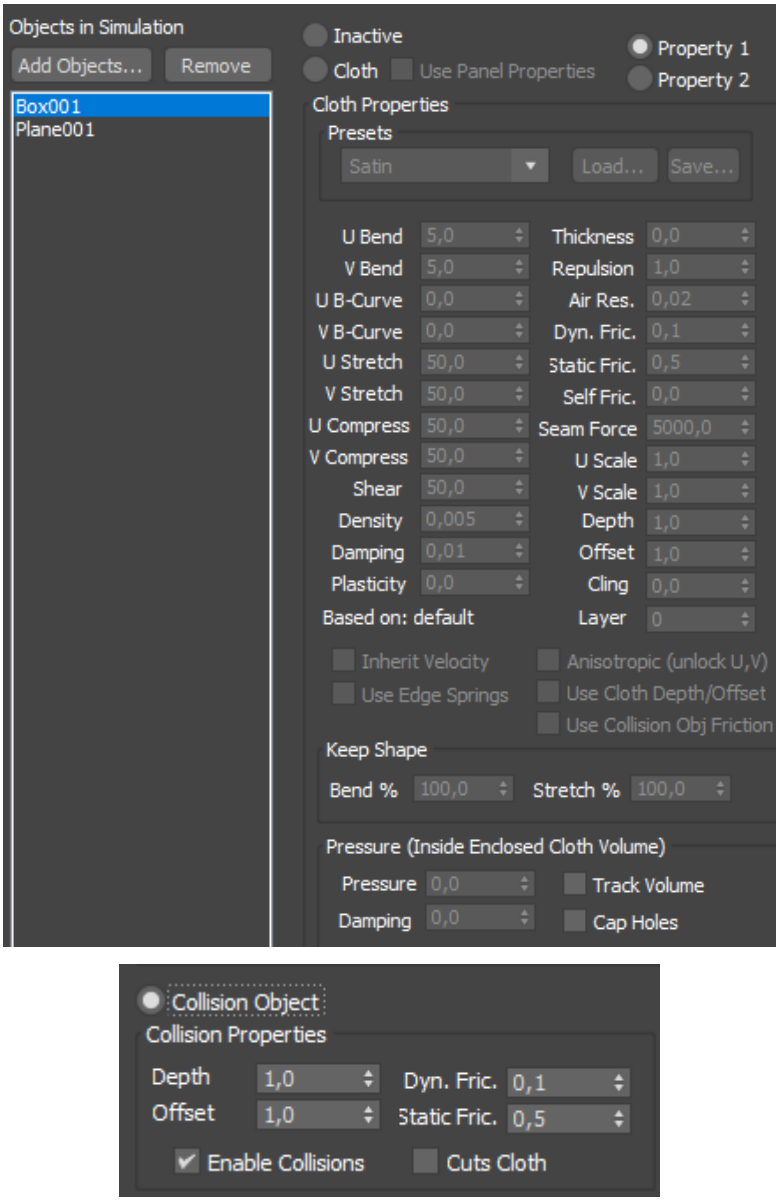
6. Prikaz i opis rezultata

U ovom dijelu rada govorit će se o tome na koji način su se kreirali modeli i scena, ali i prikazat će se programski dio koda. Također, osvrnut će se na programe u kojima je autor i sam radio te će se jednostavno opisati proces i tijek samog projekta.

6.1. Izrada i izvoz modela

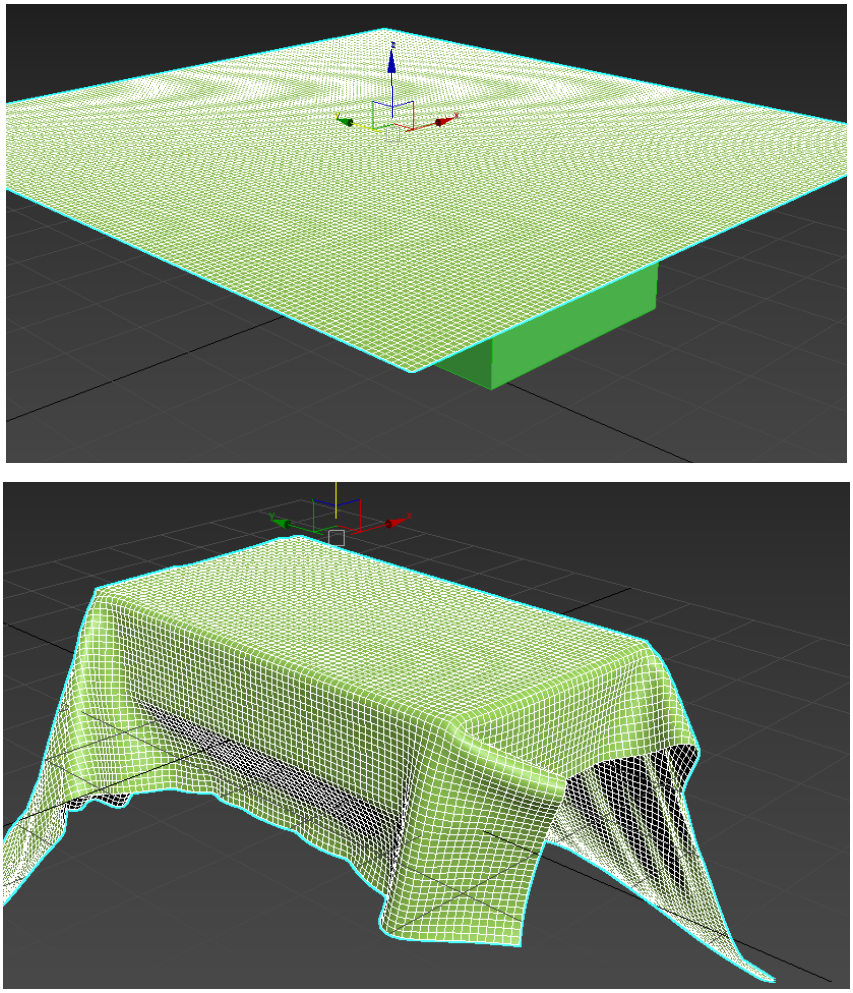
Kao što se već ranije spomenulo u ovom radu, za modeliranje 3D modela najviše se koristio program 3ds Max koji je besplatan za sve studente svijeta i to uz predočenje dokumenta studiranja (u mom slučaju studentska iksica i potvrda). Samo za model studenta koristio se Blender jer ima način rada za „*sculpting*“ što mi je bilo od velike pomoći prilikom izrade modela studenta. Kako je sam logotip fakulteta prilično jednostavnog dizajna modeliranje se počelo od šesterokuta (2D) kojemu se dodala visinu i širinu te ga se postepeno nadograđivalo. Na isti način se napravila mrežu unutar okvira šesterokuta i elemenata koje čine same logotipe. Naravno, koristila se referenca koju se pronašlo na internetu i web stranici fakulteta. Nakon modeliranja obojali su se modeli standardnim 3ds Max materijalima. Točne nijanse boje saznali su se tako što su se referentne slike otvorile u *Photoshop* programu pomoću *Eyedropper* alata. Logotipe su se centrirali u sceni pomoću centriranja pivota kako bi bili uvijek na sredini scene prilikom ubacivanja u Three.js biblioteku. Logotipi nisu memorijski zahtjevni jer su *low-poly* te nema potrebe za nečim detaljnijim. Logotipe se izvezlo u .GLB formatu jer se željelo u jednoj datoteci imati sve podatke o modelu. Na kraju ih se stavilo u projekt u direktorij pod imenom „models“. Na otprilike isti način modelirali su se i ostali modeli koji se nalaze u sceni poput stola, police, rakete, cvijeta, knjiga, poda itd.

Prilikom modeliranja tepiha koji se simulira preko okruglog poda, u programu 3ds Max, koristio se *modifier* koji se naziva „*cloth*“. Ideja je bila da se obični *plane* pomoću *modifiera* „*Toorbo smooth*“ podijeli na jednake veličine, poput mreže, te ju se stavi iznad poda. Uključi se *modifier* „*cloth*“ te se postavi da je *plane* objekt nad kojim se vrši *modifier* „*cloth*“ te se upisuju željeni brojevi za materijal koji će *cloth* simulirati. Nakon toga dodao se objekt poda i za njega se stavilo da je „*collision objekt*“. Nakon toga pokrene se simulacija na tipku „*simulate local*“. Kada je simulacija bila gotova na model tepiha primijenio se *modifier* „*editable poly*“ kako bi se mogao primijeniti određeni materijal.



Slika 6.1. Primjer kako koristiti „*cloth*“ *modifier*.

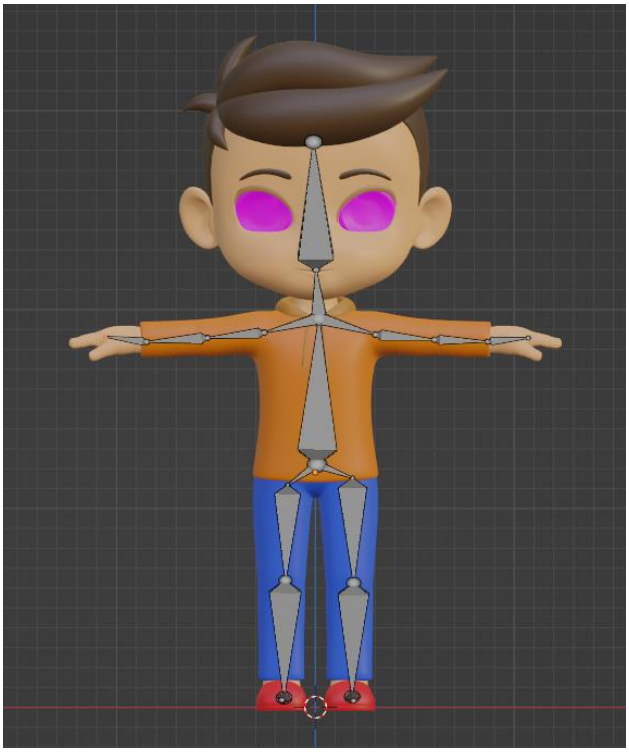
„*Box001*“ predstavlja „*collision object“*, dok je Plane001 podijeljen na manje dijelove i simuliran putem „*collision objecta“* sa materijalom „*Satin*“.



Slika 6.2. Prije i nakon korištenja *„cloth modifiera“*.

Prilikom modeliranja 3D modela studenta koristio se Blender jer u njemu postoji način modeliranja sa kistovima, takozvani *Sculpting* način rada. Sa izradom modela krenulo se isto kao i sa ostalim modelima, od same skice gdje se pomoću referentnih slika i verzije sa malo poligona došlo do one verzije sa više poligona. *Sculpting* se najviše koristio kako bi se na modelu bolje prikazale prirodne karakteristike, poput prstiju na rukama, ušiju i sl. te se samim time unaprijedio izgled modela. Nakon što se završilo s modeliranjem, modelu se moralo pomoću *modifiera* „*Armature*“ postaviti kosti kako bi se bilo u mogućnosti animirati ga tj. dati mu neku pozu (držanje olovke). Taj postupak zove se *Rigging*. Pomoću njega našem modelu dodaju se digitalne kosti te im se mijenja položaj u 3D prostoru, dok model prati kosti i rezultira određenom pozom, odnosno animacijom. U suprotnom, bi se moralo

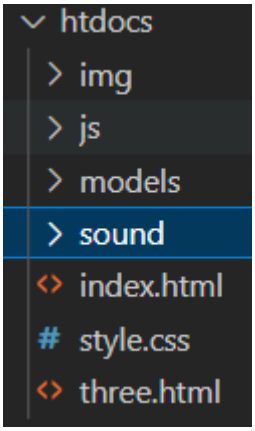
za svaku animaciju ručno pomjerati poligone modela - što bi bio jako dug i mukotrpan proces te bi se na kraju, za svaku animaciju morao raditi poseban model.



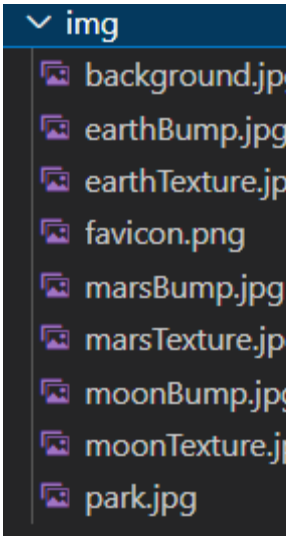
Slika 6.3. Primjer stavljanja kostiju u model (Blender).

6.2. Struktura projekta

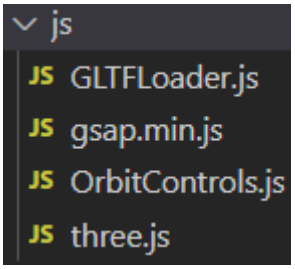
Način na koji su se prezentirali sami modele u Three.js biblioteci je kroz korištenje lokalne datoteke three.js (postupak se opisao u poglavlju 5.1.2.).



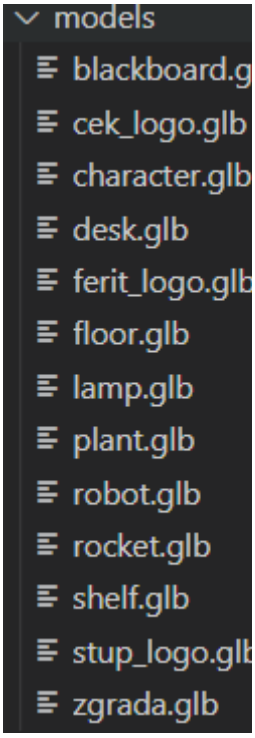
Struktura projekta



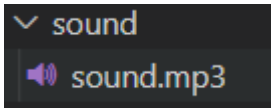
Direktorij „img“



Direktorij „js“



Direktorij „models“



Direktorij „sound“

Tablica 6.4. Struktura projekta u programu *Visual Studio Code*.

Svaki tip datoteke ima svoj direktorij. U direktoriju „img“ spremljena je „favicon“ ikona, pozadinska slika scene te teksture (*normal* + *color*) od određenih modela. U „js“ direktoriju nalazi se lokalna dadoteka „three.js“ te „OrbitControls.js“ koja služi za kontrole prilikom interakcije korisnika s modelima u sceni. Sami 3D modeli nalaze se u .GLB formatu u direktoriju „models“. Nakon modela u direktoriju „sound“ imamo zvučni zapis u .mp3 formatu koji se aktivira kada mijenjamo pogled na različite modele u izborniku. Glavne datoteke („index.html“ , „three.html“ i „style.css“) nalaze se zajedno u *rootu* projekta.

6.3. HTML struktura

```
<head>
    <meta charset="utf-8">
    <link rel="icon" href="/img/favicon.png">
    <link rel="stylesheet" href="style.css">
    <title>Tomislav Slukan - završni rad (praktični dio)</title>

    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link href="https://fonts.googleapis.com/css2?family=Raleway:wght@300&display=swap" rel="stylesheet">
</head>
```

Slika 6.5. „*Glava*“ svih HTML datoteka.

U samom zaglavlju index.html dokumenta nalaze se meta podatci za kodiranje znakova u UTF-8 formatu. Naslov projekta vidljiv je unutar <title> tagova. Ispod toga, a unutar prvog <link> taga imamo favicon u .PNG formatu - to je mala ikonica koja se pojavljuje zajedno sa naslovom projekta na kartici u web pregledniku. Pomoću drugog <link> taga povezalo se style.css iz drugog direktorija, a unutar trećeg <link> taga nalazi se hiperveza na Google font koji se koristio u cijelom projektu (*Noto Sans*, sans serif).

```
<script src="js/three.js"></script>
<script src="js/OrbitControls.js"></script>
<script src="js/GLTFLoader.js"></script>
<script src="js/gsap.min.js"></script>
```

Slika 6.6. Veze na skripte koje koristim u projektu.

Nakon otvaranja <body> tagova u .html datotekama potrebno je povezati skripte koje koristimo za Three.js. Ovdje postoje četiri skripte: prva je zaslužna za Three.js, druga za kontrole miša (zumiranje, okretanje, let oko modela), treća predstavlja GLTFLoader čiji je zadatak učitavanje .glb modela u scenu te posljednja je zadužena za glatko kretanje kamere kada na sceni prebacujemo pogled s jednog modela na druge modele. *GSAP* predstavlja *JavaScript* biblioteku koja služi za animiranje. Koristilo ju se za animiranje pokreta kamere. Više o ovim skriptama može se pročitati u poglavlju „Priprema i postavljanje scene“.

6.4. Stil (CSS)

U „style.css“ dokumentu nalazi se stil koji opisuje sve .html datoteke tj. daje izgled samoj web stranici. Koristio se responzivan dizajn tj. mjerne jedinice poput postotaka ili rem. Rem („*root element*“) je relativna mjera u odnosu na veličinu fonta koja predstavlja osnovni element, odnosno <body>. Omogućava skaliranje prema potrebi te nema ne željnih rezultata prilikom prikaza web stranice na različitim rezolucijama. 1rem iznosi 16px.

6.5. Priprema i postavljanje scene

U biblioteci Three.js prije nego što se postave sami modeli u scenu, potrebno je inicijalizirati istu. Three.js nam je tu olakšao jer je vrlo jednostavno - u samo jednoj liniji koda može se inicijalizirati scena. Osim inicijalizacije scene potrebno je na istu dodati kameru, osvjetljenje te renderer koji će sve to ispisati na ekran korisnika.

```
//scene setup
var scene = new THREE.Scene();
```

Slika 6.7. Inicijalizacija same scene u Three.js-u.

```
//scenebackground image
const bgLoader = new THREE.TextureLoader();
scene.background = bgLoader.load( './img/background.jpg' );
```

Slika 6.8. Dodavanje pozadinske slike scene.

6.5.1. Kamera

```
//perspective camera and adjustment
var camera = new THREE.PerspectiveCamera( 80, window.innerWidth / window.innerHeight, 0.1, 1000 );
camera.position.set(0, 10, 15);
scene.add(camera);
```

Slika 6.9. Perspektivna kamera.

Odabrala se perspektivna kamera jer je prirodna i rađena je prema ljudskom oku. Za vidno polje odabralo se 80. Nije se koristilo platno (eng. *canvas*), već se koristila širina i dužina cjelokupnog prozora - to je omjer nakon vidnog polja. Sama pozicija kamere stavila se jako blizu samih modela - to govori broj 0.1 (zadana vrijednost je 1). Leća kamere od udaljenije „*far*“ plohe udaljena je za 1000 (zadana vrijednost je 2000). Nakon pozicioniranja kamere na određene koordinate u sceni ona se jednostavno dodala na istu.

```
document.getElementById('ferit_logo').addEventListener('click', (event) => {

    gsap.to( camera, {
    duration: 2,
    zoom: 3,
    onUpdate: function () {

        camera.updateProjectionMatrix();

    }
} );

gsap.to( controls.target, {
    duration: 2,
    x:-2.5, y: 4, z: -2.1,
    onUpdate: function () {

        controls.update();

    }
} );

    var audio = new Audio('sound/sound.mp3');
    audio.play();
});
```

Slika 6.10. Glatki prijelazi kamere ostvaren pomoću GSAP-a.

Za svaki model koji se nalazi u glavnom izborniku scene definiran je glatki prijelaz kamere s jednog mjesta na drugo. Efekt daje bolje korisničko iskustvo. Za animaciju kamere koristio se *GSAP*. *GSAP* nije dio Three.js biblioteke već predstavlja posebnu JavaScript biblioteku koja se koristi za animaciju u modernom web-u vrlo visokih performansi.

Model se odabrao pomoću ID-a te se stavio slušatelj na odabir lika modela iz glavnog izbornika. Kada se klikne na neki link modela - kamera pomoću *GSAP-a* definira na koje koordinate će kamera otići, koliko dugo će put do tamo trajati te samo uvećanje. Nakon toga mora se ažurirati matrica kamere, tj. položaj, kako bi mogli pregledati model (ukratko, stavljanje *gizmo* modela u centar).

Nakon ažuriranja koordinata kamere pomoću *GSAP* biblioteke dodao se zvuk u .mp3 formatu koji se pokreće pomoću JavaScripta i funkcije *audio.play()*.

6.5.2. Izvori svjetlosti

```
//three lights in colours
const light = new THREE.PointLight( "orange", 10, 25 );
light.position.set( -15, 20, 0 );
scene.add( light );

const light2 = new THREE.PointLight( "purple", 10, 25 );
light2.position.set( 15, 10, 0 );
scene.add( light2 );

const light3 = new THREE.PointLight( "blue", 10, 25 );
light3.position.set( 0, 10, -10 );
scene.add( light3 );
```

Slika 6.11. Izvori svjetlosti koji su se koristili u lampi.

Točkasti izvor svjetlosti različitih boja (narančasta, ljubičasta i plava) koji se nalaze na koordinatama samostojeće lampe u sceni. Svi izvori imaju isti intenzitet od 10 te distancu koje svjetlo pokriva od 25.

```
document.addEventListener('keydown', (event) => {
    if(event.keyCode == 49){
        lampLight.color.set("purple");
    }
    else if(event.keyCode == 50){
        lampLight.color.set("red");
    }
    else if(event.keyCode == 51){
        lampLight.color.set("yellow");
    }
    else if(event.keyCode == 52){
        lampLight.color.set("blue");
    }
    else if(event.keyCode == 107){
            lampLight.distance += 1;
            if(lampLight.distance >= 15){
                lampLight.distance = 15;
            }
    }
    else if(event.keyCode == 109){
        lampLight.distance -= 1;
            if(lampLight.distance <= 1){
                lampLight.distance = 1;
            }
    }
});
```

Slika 6.12. Promjena boje i intenziteta svjetlosti na samostojećoj lampi u sceni pomoću tipkovnice.

Dodao se slušatelj na pritisak tipke na tipkovnici. Ako je pritisnuta tipka od 1 do 4 - mijenja se boja svjetla, dok pritisak plusa ili minusa povećava, odnosno smanjuje intenzitet svjetla odabrane boje. *KeyCode* predstavlja ASCII vrijednost tipke na tipkovnici.

```
//spot light - light for shadows
var spotLight = new THREE.SpotLight("#fff", 2);
spotLight.castShadow = true;
spotLight.shadow.bias = -0.0001;
spotLight.shadow.mapSize.width = 1024*4;
spotLight.shadow.mapSize.height = 1024*4;
scene.add(spotLight);
```

Slika 6.13 Izvor svjetlosti koji se koristio za bolju vidljivost kamere (detaljno objašnjeno u odlomku „Animacija“).

SpotLight je izvor svjetlosti koje se emitira iz jedne točke u jednom smjeru, duž stošca koji se povećava što je dalje od svjetla. Definirana je bijela boja, intenziteta 2. Također je omogućena projekcija sjene. *Shadow.bias* govori koliko dodati ili oduzeti od normalizirane dubine pri odlučivanju je li površina u sjeni. Zadana vrijednost je 0, ali ovdje vrlo male prilagodbe mogu pomoći u smanjenu efekta artefakata u sjenama. Za veličinu mape sjene koristio se 1024*4 za širinu i visinu jer - što je vrijednost veća to će biti kvalitetnije sjene, ali duže vrijeme računanja. Zadana vrijednost je 512*512.

6.5.3. Renderer

```
//renderer and sizing setup
var renderer = new THREE.WebGLRenderer({antialias: true, precizion: "highp", powerPreference: "high-performance"});
renderer.setSize( window.innerWidth, window.innerHeight );
document.body.appendChild( renderer.domElement );
renderer.toneMapping = THREE.ReinhardToneMapping;
renderer.toneMappingExposure = 2.3;
renderer.shadowMap.enabled = true;
renderer.shadowMap.type = THREE.PCFSoftShadowMap;
```

Slika 6.14. WebGLRenderer.

Za parametre koje prima konstruktor WebGL renderera naveo se *antialias „true“* kako bi se dobili glatki rubove na 3D modelima u sceni. Preciznost sjenčara stavio se na *highhp* te postavke performanse na *visoko*. Ovisno o WebGL kontekstu pokazuje koja je grafička konfiguracija prikladna za isti.

Za veličinu *canvasa* uzima se u obzir omjer piksela uređaja, a to je ujedno cijeli prozor web preglednika.

Nakon definiranja omjera prozora, potrebno je definirati na koje platno će se iscrtati izlaz. Ako u konstruktoru nije navedeno platno direktno, Three.js će automatski stvoriti renderer u konstruktoru pomoću linije koda u nastavku:

```
document.body.appendChild( renderer.domElement );
```

Slika 6.15. Indirektno definiranje platna.

ToneMapping je tehnika koja se koristi u računalnoj grafici te obradi slike koja preslikava jedan skup boja u drugi kako bi se približio izgled slika visokog dinamičkog raspona u mediju koji ima ograničeni dinamički raspon poput CRT ili LCD monitora, projektora i sl. Tonsko mapiranje rješava problem snažnog smanjenja kontrasta - od sjaja scene do raspona koji se može prikazati dok istovremeno čuva detalje i izgled boje koji su važni za originalni sadržaj scene.

Na rendereru se omogućilo mapiranje sjena na sceni te se koristio *PCGSoftShadowMap*. Isti filtrira mape sjena koristeći PCF, odnosno *Percentage-Closer* algoritam za filtriranje. Ideja je uzorkovati mapu sjene oko trenutnog piksela te usporediti njegovu dubinu sa svim uzorcima. Usrednjavanjem rezultata dobije se glađa liniju između svjetla i sjene.

6.5.4. Učitavanje 3D modela

```
//GLTF loader setup
var loader = new THREE.GLTFLoader(loadingManager);
```

Slika 6.16. GLTF *loader* potreban za učitavanje 3D modela na scenu u .GLB formatu.

```
//ferit_logo
var ferit_logo = loader.load("models/ferit_logo.glb", function (gltf) {
    ferit_logo = gltf.scene;
    ferit_logo.traverse( function ( ferit_logo ) {
        if ( ferit_logo.isMesh ){
            ferit_logo.castShadow = true;
            ferit_logo.receiveShadow = true;
        }
    });
    ferit_logo.position.set(-2.5, 4, -2.1);
    scene.add(ferit_logo);
});
```

Slika 6.17. Primjer učitavanja *ferit_logo.glb* modela u scenu pomoću *loadera*.

Pomoću .gltf *loadera* svaki model potrebno je postaviti na scenu. U ovom slučaju svi su modeli bili u .GLB formatu. Ukoliko se model sastoji od više odvojenih dijelova potrebno je pozvati *traverse* funkciju i dok je uvjet model.isMesh = *true* model neće imati problema sa projekcijom sjene u sceni.

Svaki se 3D model posebno pozicionirao u istu scenu te ih se pomaknuo za određenu vrijednost od ishodišta. Pozicija samih modela ovisi i o dijelu iz 3ds Max-a. Da se nije njihov pivot stavio na sredinu, modeli bi za tu razliku bili odmaknuti od ishodišta u bilo kojem programu ili 3D pregledniku, kao i u Three.js biblioteci. Iz tog razloga je važno prije samog izvođenja modela iz programa za modeliranje paziti da se pivot modela nalazi točno u srediti.

```
//earth
const earthGeometry = new THREE.SphereGeometry( 1, 32, 32 );
const earthTexture = new THREE.TextureLoader().load( './img/earthTexture.jpg' );
const earthBumpTexture = new THREE.TextureLoader().load( './img/earthBump.jpg' );
const earthMaterial = new THREE.MeshPhongMaterial( { map: earthTexture,
    bumpMap: earthBumpTexture, bumpScale: 0.01} );
const earth = new THREE.Mesh( earthGeometry, earthMaterial );
earth.position.set(8, 1, 2);
earth.rotation.set(-12, 0, -12);
earth.castShadow = true;
earth.receiveShadow = true;
scene.add(earth);
```

Slika 6.18. Primjer učitavanja Three.js gotovih 3D oblika, u ovom slučaju sfere s teksturom Zemlje.

Osim ovih 3D modela u biblioteci Three.js moguće je dodavati gotove oblike te im dati određene teksture u vidu bitmapa. Za taj dio koristile su se tri kugle koje predstavljaju Zemlju, Mjesec i Mars te imaju različite teksture. Za primjer Zemlje koristile su se *sphereGeometry* velilčine 1 te 32 horizontalna i vertikalna segmenata. Za teksture koristila se boja i normal mapa Zemlje koju je autor sam pronašao na internetu te ih stavio u *MeshPhongMaterial* s još jednim dodatnim parametrom, a to je *bumpScale* što predstavlja povećanje detalja na normal mapi. Nakon definiranje oblika i tekstura, stavile su se koordinate pozicije u sceni, omogućilo se primanje i distribucija sjena te konačno isti oblik dodao se na samu scenu. Za modele Mjeseca i Marsa koristio se potpuno isti proces.

6.5.5. Responzivnost

```
//responsive canvas resize
window.addEventListener('resize', function(  )
{
    var width = window.innerWidth;
    var height = window.innerHeight;
    renderer.setSize( width, height );
    camera.aspect = width / height;
    camera.updateProjectionMatrix();
});
```

Slika 6.19. Ostvarenje responzivnosti prilikom promjene veličine web preglednika.

Prilikom korisnikove promjene veličine zaslona, računa se omjer trenutne veličine širine i visine ekrana kako bi cijela scene bila uvijek u zadanom omjeru.

6.5.6. Kontrole

```
//orbit controls setup
var controls = new THREE.OrbitControls(camera, renderer.domElement);
controls.maxPolarAngle = Math.PI / 2;
controls.enablePan = false;
controls.enableDamping = true;
controls.dampingFactor = 0.5;
```

Slika 6.20. Ostvarivanje interakcije korisnika sa 3D modelima na sceni.

Na ovaj način korisniku je omogućena interakcije sa 3D modelima te može koristiti kontrole na mišu (primarni i sekundarni klik) te kontrole na tipkovnici (SHIFT, CTRL) za pozicioniranje modela i kamere u sceni. Pomoću ove linije koda korisniku dajemo bolje iskustvo prilikom korištenja web stranice.

MaxPolarAngle se odnosi na to koliko daleko možemo okomito kružiti oko modela - gornja granica. Raspon je od 0 do Math.PI radijana. Isti se podijeli sa brojem 2 jer se ne želi da korisnik kamerom može vidjeti modele od ispod.

Na autorovim kontrolama nije se omogućio *pan*, ali se mogućio *dampingFactor* koji iznosi 0.5. On pruža inerciju - što pridonosi osjećaju težine nad kontrolama.

6.5.7. Pozadina

```
//background image (canvas) setup
var img = new Image();
img.onload = function () {
    scene.background = new THREE.TextureLoader().load(img.src);
    setBackground(scene, img.width, img.height);
};
img.src = "img/background.png";
```

Slika 6.21. Promjena pozadinske fotografije u sceni po izboru.

Promjena pozadinske fotografije koja se nalazi u direktoriju „img“ te je .png formata. Kada se pozadinska slika učita pokreće se funkcija koja na scenu (pozadinu) dodaje bitmapu iz određenog izvora.

6.5.8. Animacija

```
//every new frame function "render" is called
function render(){
    requestAnimationFrame(render);

    if(ferit_logo && stup_logo && cek_logo && character){
        ferit_logo.rotation.y += 0.01;
        stup_logo.rotation.y += 0.01;
        cek_logo.rotation.y -= 0.01;
        rocket.rotation.y -= 0.01;
        mars.rotation.z += 0.01;
        moon.rotation.x += 0.01;
        earth.rotation.y -= 0.01;

        spotLight.position.set(
            camera.position.x + 10,
            camera.position.y + 10,
            camera.position.z + 10
            );
    }

    renderer.render( scene, camera );
}
```

Slika 6.22. „Render()“ funkcija koja služi za prikaz animacije objekata na sceni.

Funkcija *requestAnimationFrame* nakon svake promjene *framea* pozove funkciju render() koja osvježi podatke vezane za rotaciju svakog objekta koji se nalazi u sceni.

Osim bilježenja svake promjene rotacije objekta na sceni u ovu funkciju dodao se *spotLight* (čiji je dio dodatno objašnjen). Ideja je da se stavi svjetlo ispred kamere te na taj način nijedan objekt u sceni neće biti u potpunom mraku ili ne vidljiv korisniku. Sve tri koordinate *spotLighta* stavilo se na poziciju kamere te se raspršilo tu svjetlost za vrijednost 10 (i to za svaku koordinatu).

Nakon toga, na kraju, poziva se renderer - koji kao argumente prima scenu i kameru - iz istog razloga (osvježavanje scene nakon promjene *framea*).

6.5.9. Traka za napredak (učitavanje scene)

```
<div class="progress-bar-container">
    <label for="progress-bar" id="progress-bar-label">Scene is loading</label>
    <progress id="progress-bar" value="0" max="100"></progress>
</div>

<script>
    var progressBar = document.getElementById('progress-bar');
    var progressBarContainer = document.querySelector('.progress-bar-container');
    var loadingManager = new THREE.LoadingManager();
        loadingManager.onProgress = function(url, itemsLoaded, itemsTotal){
            progressBar.value = itemsLoaded / itemsTotal * 100;
        }
        loadingManager.onLoad = function(){
            progressBarContainer.style.display = 'none';
        }
    </script>
```

Slika 6.23. Slika prikazuje na koji način je napravljena komponenta trake za učitavanje cijele scene.

Traka za učitavanje vrlo je važna komponenta korisničkog iskustva jer korisniku daje procjenu vremena koliko mora čekati te što se događa sa web stanicom.

U HTML-u vidimo klasu koja u sebi sadrži <label> i <progress> sa minimalnim i maksimalnim vrijednostima. ID služi za dizajn u CSS-u te dohvaćanje elementa u JavaScriptu.

Pomoću JavaScripta dohvatila se klasa i <progress> - pomoću njihovog ID-a te se inicijalizirao *loadingManager* koji je dio Three.js-a. Ukratko, *loadingManager* ovdje sadrži dvije funkcije, a to su: *onLoad* i *onProgress.* Prva funkcija, *onProgress*, koristi parametre poput URL-a *itema* te koliko je ukupno i koliko se *itema* trenutno učitalo.

6.5.10. Direktno definiranje platna u HTML-u te prikaz scene u istom

Kao što se već napisao ranije, ako u konstruktor renderera direktno ne navedemo platno koje koristimo za prikaz scene, Three.js će sam kreirati isti. Ukoliko želimo definirati svoje platno u koje ćemo prikazati našu scenu, proces je nešto drugačiji. Navedene slike prikazuju što se sve treba promijeniti kako bismo mogli dobiti scenu u direktno navedenom platnu HTML-a.

```
<canvas
  id="canvas"
  style="display: block; width: 100%; height: 100vh"
></canvas>
```

Slika 6.24. Definiranje HTML platna.

Definiramo platno u HTML-u uz definirani ID i stil po želji.

```
var canvas = document.getElementById("canvas");
var canvasWidth = document.getElementById("canvas").clientWidth;
var canvasHeiht = document.getElementById("canvas").clientHeight;
```

Slika 6.25. Definiranje visine i širine platna.

Isto to platno dohvati se u JavaScriptu pomoću njegovog ID-a. Nakon toga definira se širina i visina platna.

```
var camera = new THREE.PerspectiveCamera(
  80,
  canvasWidth / canvasHeiht,
  0.01,
  1000
);
camera.position.set(0, 1, 2);
scene.add(camera);
```

Slika 6.26 Inicijalizacija kamere.

Prilikom inicijalizacije kamere za atribute širine i visine uvrstimo vrijednosti iz prethodne slike.

```
//renderer and sizing setup
var renderer = new THREE.WebGLRenderer({
  canvas: canvas,
  antialias: true,
  precizion: "highp",
  powerPreference: "high-performance",
});
renderer.setSize(canvasWidth, canvasHeiht);
```

Slika 6.27. Inicijalizacija *renderera*.

Prilikom inicijalizacije renderera u njegovom konstruktoru potrebno je direktno napisati koje platno koristimo (*canvas: canvas*).

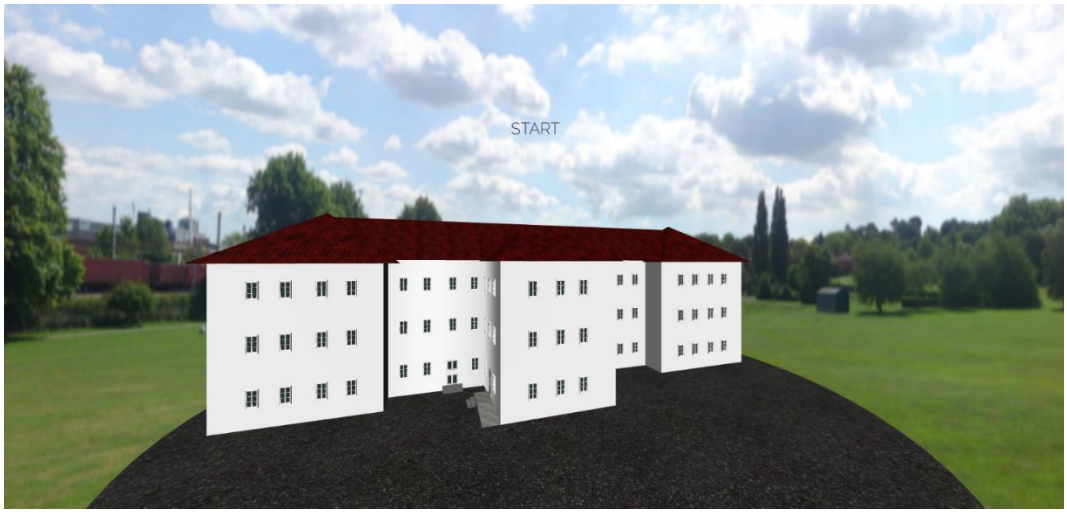
```
//responsive canvas resize
window.addEventListener("resize", function () {
  var width = canvasWidth;
  var height = canvasHeiht;
  renderer.setSize(width, height);
  camera.aspect = width / height;
  camera.updateProjectionMatrix();
});
```

Slika 6.28. Responzivnost scene.

Kako bismo zadržali responzivnost scene potrebno je u *renderer.setSize()* uvrstiti širinu i visinu platna te iste atribute širine i visine staviti u *camera.aspect*.

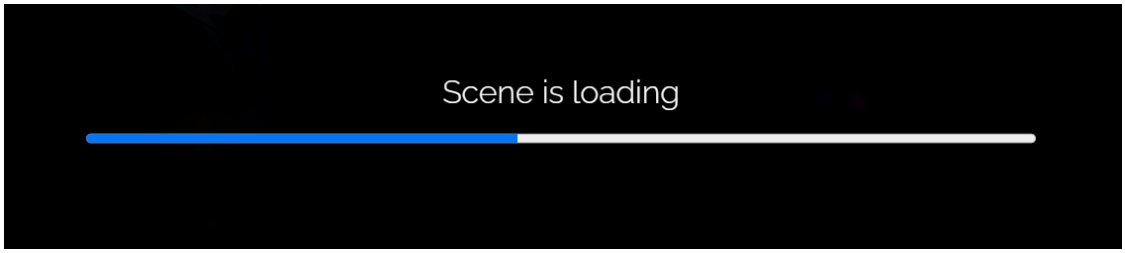
7. Izgled napravljenih rješenja

Za ovaj projekt napravljeno je više od 10 3D modela s ciljem da se stvori ugodno korisničko iskustvo istraživanja jedne web stranice. Na početnoj stranici, nakon trake za napredak, nalazi se rekreacija zgrade FERIT-a sa pozadinskom fotografijom okoliša u direktno navedenom HTML platnu. Zgrada se okreće za 360 stupnjeva dok korisnik ne pristupi „start“ tipki. Nakon toga inicijalizira se glavna scena, koja je bogatija što se tiče broja 3D modela te funkcionalnosti poput zvuka, prijelaza kamere s modela na model, interakcije korisnika i modela i sl.

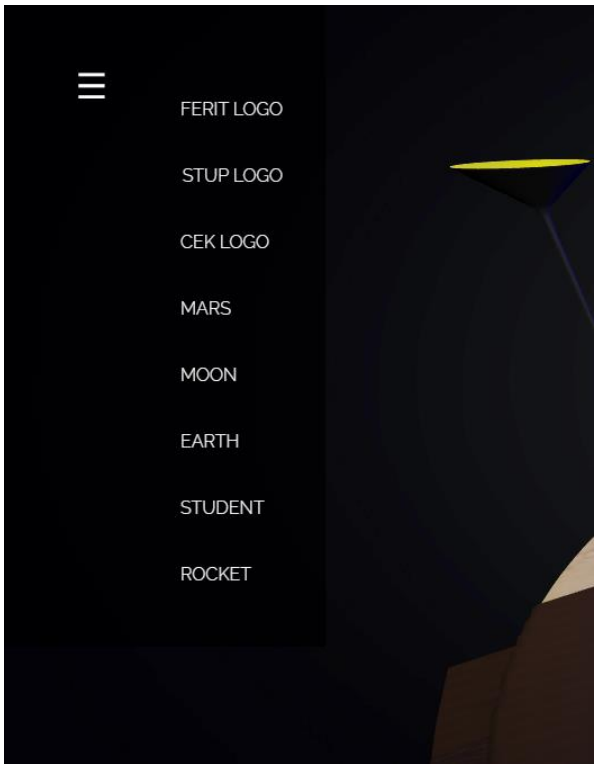


Slika 7.1. Rekreacija FERIT zgrade.

Slika 7.2. Glavna scena.

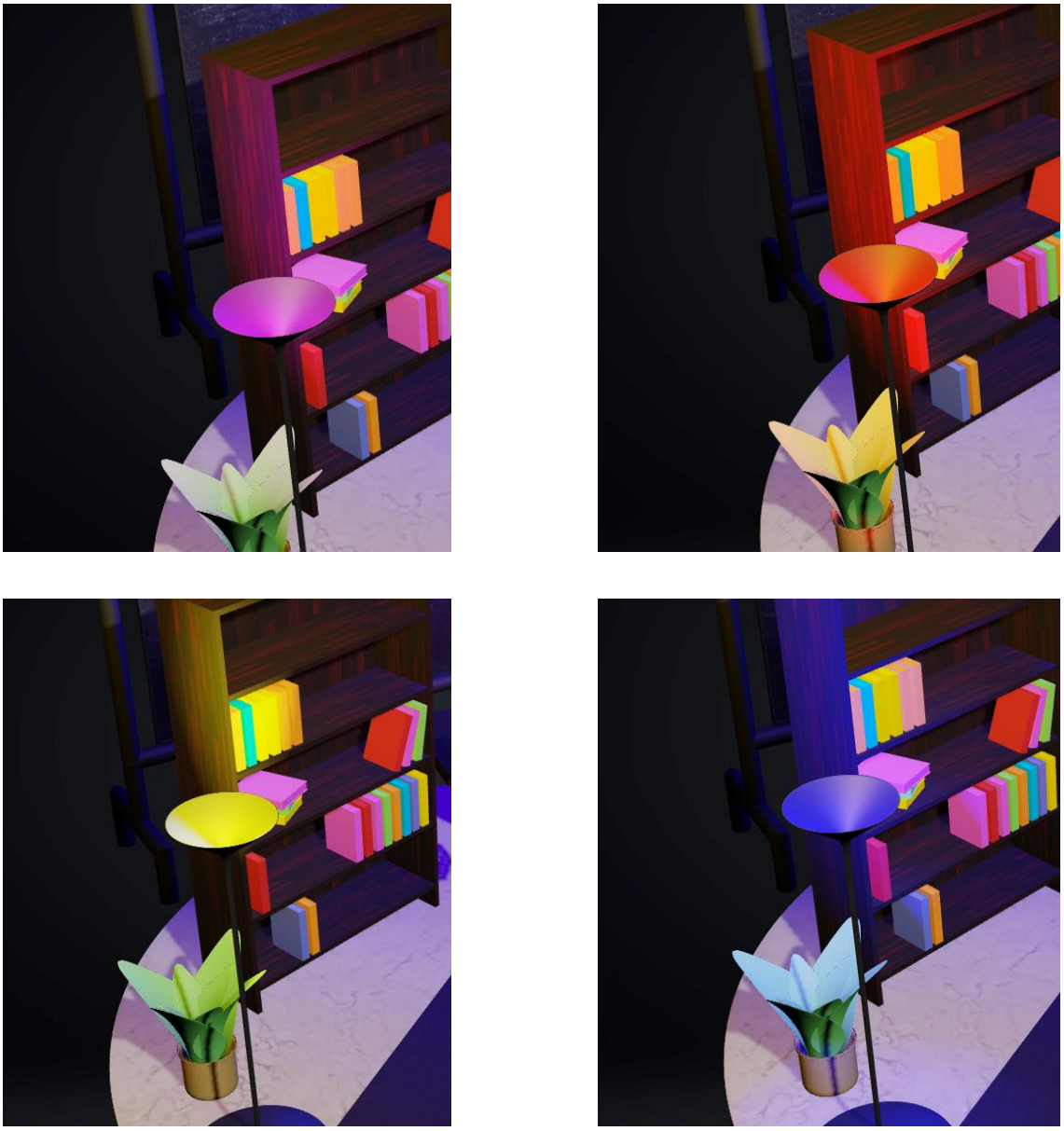


Slika 7.3. Prikaz trake za napredak.



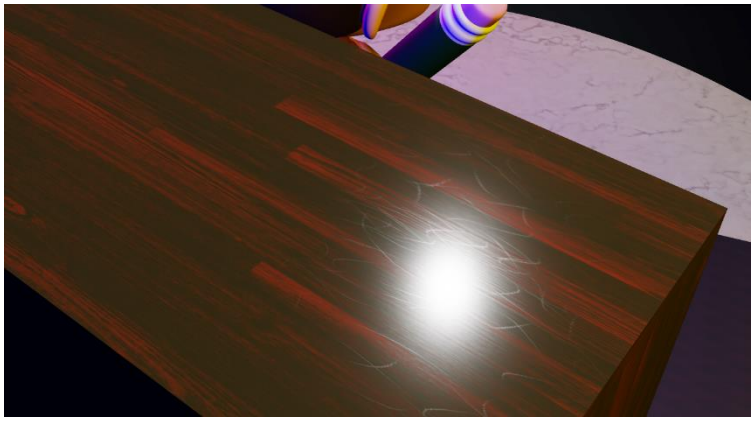
Slika 7.4. Glavni izbornik za prikaz modela scene.

Korisnik može vrlo lako, pomoću izbornika sa lijeve strane, odlučiti koji model želi pogledati detaljnije te će ga kamera odvesti do istog. Korisnik je u mogućnosti interakcije sa samim modelom.

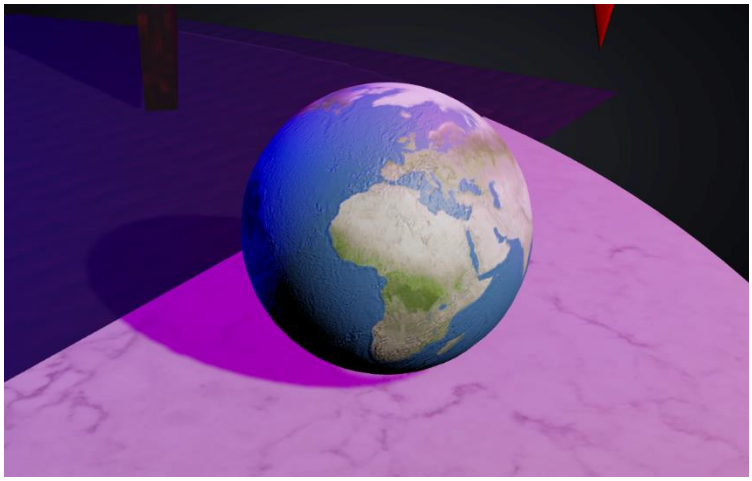


Slika 7.5. Moguće je mijenjati boju i intenzitet svjetla lampe u sceni na određene tipke na tipkovnici.

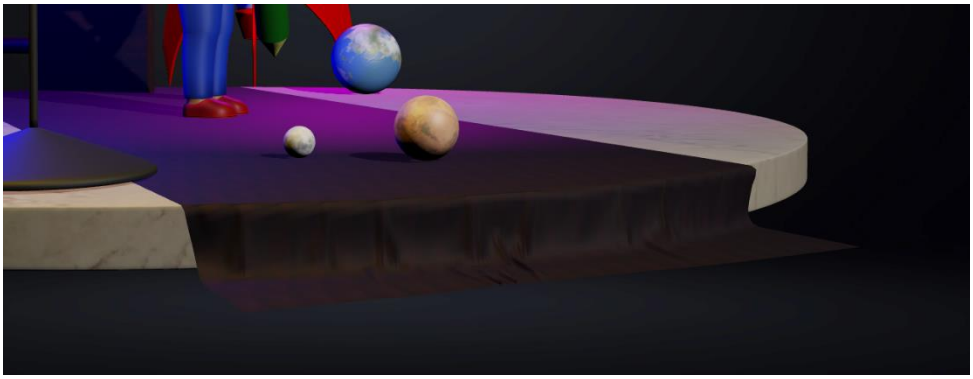
Dodatne funkcionalnosti poput promjene boje i intenziteta svjetla iz lampe na sceni korisniku daju bolje iskustvo.



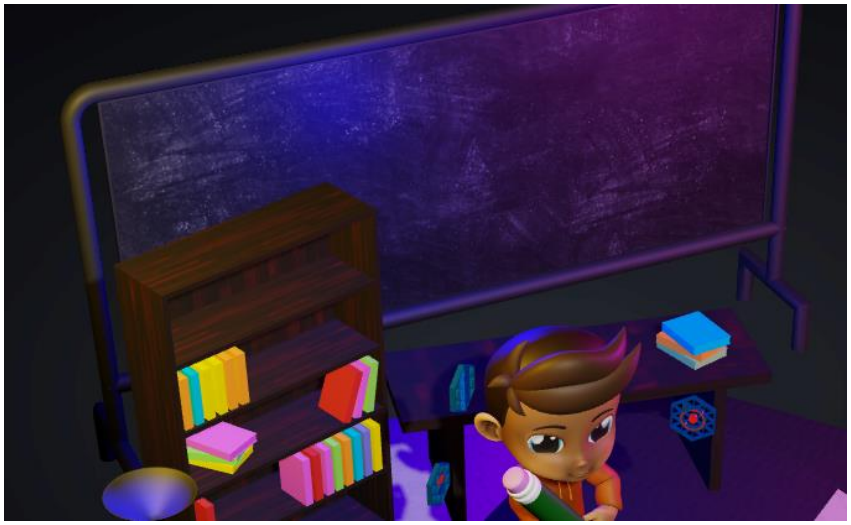
Slika 7.6. Tekstura drveta pod utjecajem svjetlosti na njegovu površinu.



Slika 7.7. Sfera (Zemlja) napravljena pomoću gotove Three.js sfere te primjena mapa boje i bump (normal mape).



Slika 7.8. 3Ds Max *cloth modifier*.



Slika 7.9. Prikaz primjenjenog materijala na ploči za pisanje.

Slika 7.10. Prikaz studenta na koji je primijenjen *rig.* Ima određenu poziciju tijela.

Navedene slike u nizu prikazuju modele iz blizine, kako bi uočili detalje na samim teksturama poput drveta, ploče, tepiha te odsjaja svjetla koje daju različiti materijali.

8. Problem sa više platna u sceni

Ukoliko bismo došli na ideju da želimo na jednoj web stranici više odvojenih platna i na njima različite sadržaje, prvo što bi nam palo na pamet bilo bi da imamo više različitih renderera za ta različita platna tj. kontekste.

Prvi problem je taj što web preglednik daje limit koliko WebGL konteksta možemo imati. Mobilni web preglednici imaju limit od osam WebGL konteksta. Ako se pak kreira deveti kontekst, prvi će biti izgubljen. Drugi problem je taj što WebGL konteksti ne mogu dijeliti resurse jedni s drugima - što znači da je potrebno inicijalizirati puno varijabli s različitim imenima i to onoliko puta koliko imamo različitih konteksta.

Rješenje ovog problema je kreirati jedno platno koje ispunjava eng. *viewport* u pozadini i neki drugi element koji će predstavljati virtualno platno. Tako možemo napraviti jedan renderer i jednu scenu za svako virtualno platno. Tada provjeravamo nalazi li se platno na zaslonu ili je nevidljiv u pozadini te ako se nalazi na zaslonu kažemo Three.js-u da na njega iscrta scenu. Tako rješavamo oba problema (problem sa više od osam platna te problem sa više konteksta i inicijaliziranja identičnih varijabli). [45]

9. Zaključak

Možemo zaključiti kako je biblioteka Three.js svestrana te ako ju želimo upotrijebiti u dogovoru sa 3D dizajnerima možemo napraviti web stranice koje ostavljaju bez daha te podižu korisničko iskustvo na višu razinu.

Three.js je biblioteka koja je definitivno budućnost razvoja web stranica jer je vrlo pojednostavljena i daje odlično korisničko iskustvo. Ako usporedimo kompleksnost WebGL-a i Three.js smatram da je Three.js biblioteka napravljena za industriju i svakako se treba još razvijati u većim zajednicama programera.

Literatura

1. https://en.wikipedia.org/wiki/Three.js#:~:text=History,Three.,ported%20to%20JavaScript%20in%202009, pristup: 20.6.2022.
2. https://codame.com/artists/mr-doob, pristup: 20.10.2022.
3. https://caniuse.com/webgl, pristup: 8.6.2022.
4. https://threejs.org/manual/#en/fundamentals , pristup: 20.10.2022.
5. https://stackshare.io/three-js , pristup: 20.10.2022.
6. https://hiteshkrsahu.medium.com/player-2-has-joined-the-game-three-js-vs-babylon-dea49bf00466#:~:text=Just%20like%20THREE%20JS%20Babylon,users%20as%20Node%20Packages%20%26%20CDN, pristup: 20.10.2022.
7. https://gamedev.stackexchange.com/questions/29338/can-browser-based-games-be-developed-without-js-flash#:~:text=However%2C%20no%2C%203D%20graphics%20in,Unity3D%2C%20Silverlight%2C%20etc), pristup: 20.10.2022.
8. https://gamedev.stackexchange.com/questions/29338/can-browser-based-games-be-developed-without-js-flash#:~:text=However%2C%20no%2C%203D%20graphics%20in,Unity3D%2C%20Silverlight%2C%20etc), pristup: 20.10.2022.
9. https://enterprise.arcgis.com/en/system-requirements/10.5/windows/scene-viewer-requirements.htm#:~:text=reopen%20your%20browser.-,Hardware%20requirements,512%20MB%20of%20video%20memory, pristup: 20.10.2022.
10. https://threejs.org/docs/#manual/en/introduction/How-to-create-VR-content, pristup: 20.10.2022.
11. https://medium.com/sopra-steria-norge/get-started-with-augmented-reality-on-the-web-using-three-js-and-webxr-part-1-8b07757fc23a, pristup: 20.10.2022.
12. https://www.techtarget.com/whatis/definition/3D-model, pristup: 20.10.2022.
13. https://cgcookie.com/posts/6-principles-of-great-3d-modeling, pristup: 20.10.2022.
14. https://www.pinterest.com/pin/770467448734085202/, pristup: 20.10.2022.
15. https://www.makeuseof.com/primitives-in-3d-modeling/, pristup: 20.10.2022
16. https://arsenal.cgtrader.com/blog/difference-between-high-poly-and-low-poly-models, pristup: 20.10.2022.

17. https://www.researchgate.net/figure/Left-3D-Model-with-and-without-texture-mapping-Right-2D-texture-Image-taken-from-7_fig5_301818620, pristup: 20.10.2022.
18. https://www.makeuseof.com/textures-vs-materials-3d-modeling/, pristup: 20.10.2022.
19. https://cgtricks.com/basic-guide-for-testures-map-types-yanko-stefanov/,pristup: 20.10.2022.
20. https://conceptartempire.com/uv-mapping-unwrapping/, pristup: 21.10.2022.
21. https://www.linkedin.com/pulse/shaders-short-story-binigya-dahal, https://en.wikipedia.org/wiki/Shader , pristup: 21.10.2022.
22. https://web.cs.wpi.edu/~matt/courses/cs563/talks/shadow/shadow.html, https://www.geeksforgeeks.org/z-buffer-depth-buffer-method/, pristup: 21.20.2022.
23. https://substance3d.adobe.com/documentation/bake/what-is-baking-172818449.html, pristup: 21.10.2022.
24. https://docs.fileformat.com/3d/gltf/, https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON#:~:text=JavaScript%20Object%20Notation%20(JSON)%20is,page%2C%20or%20vice%20versa)., https://help.sketchfab.com/hc/en-us/articles/360046421631-glTF-GLB-and-USDZ, pristup: 21.10.2022.
25. https://docs.fileformat.com/3d/obj/, pristup: 21.10.2022.
26. https://www.marxentlabs.com/fbx-files/#:~:text=An%20FBX%20(.,(AR%2FVR)%20development., pristup: 21.10.2022.
27. https://threejs.org/docs/index.html?q=scen#manual/en/introduction/Installation, pristup: 22.10.2022.
28. https://threejs.org/docs/?q=scene#api/en/scenes/Scene, pristup: 22.10.2022.
29. https://threejs.org/docs/#api/en/lights/PointLight, pristup: 22.10.2022.
30. https://www.tutorialspoint.com/javafx/point_spot_effect.htm, pristup: 20.10.2022.
31. https://threejs.org/docs/#api/en/lights/DirectionalLight, pristup: 22.10.2022.
32. https://docs.unity3d.com/Manual/Lighting.html, pristup:, pristup: 20.10.2022.
33. https://threejs.org/docs/#api/en/lights/AmbientLight, pristup: 22.10.2022.
34. https://openframeworks.cc/documentation/gl/ofLight/, pristup: 20.10.2022.
35. https://threejs.org/docs/#api/en/lights/HemisphereLight, pristup: 20.10.2022.

36. https://www.researchgate.net/figure/Illustration-of-a-simplified-model-of-near-light-where-a-hemisphere-of-radius-r-is_fig2_221110527, pristup: 22.10.2022.
37. https://threejs.org/docs/#api/en/cameras/OrthographicCamera, pristup: 22.10.2022.
38. https://stackoverflow.com/questions/36573283/from-perspective-picture-to-orthographic-picture, pristup: 22.10.2022.
39. https://threejs.org/docs/#api/en/cameras/PerspectiveCamera, pristup: 22.10.2022.
40. https://stackoverflow.com/questions/36573283/from-perspective-picture-to-orthographic-picture, pristup: 22.10.2022.
41. https://threejs.org/docs/?q=loading#manual/en/introduction/Loading-3D-models, pristup: 22.10.2022.
42. https://threejs.org/docs/?q=trans#manual/en/introduction/Matrix-transformations, pristup: 22.10.2022.
43. https://threejs.org/docs/?q=render#api/en/renderers/WebGLRenderer, pristup: 22.10.2022.
44. https://threejs.org/docs/?q=draco#examples/en/loaders/DRACOLoader, pristup: 22.10.2022.
45. https://r105.threejsfundamentals.org/threejs/lessons/threejs-multiple-scenes.html, pristup: 4.12.2022.

Sažetak

U ovom diplomskom radu, čija je tema upotreba Three.js biblioteke za prikazivanje 3D sadržaja u web preglednicima, saznali smo što je potrebno za izradu samih 3D modela te kako ih prezentirati na web-u. Također, može se iščitati koji programi se trebaju koristiti, te na što se treba obratiti pažnja prilikom izrade istih.

Kako bi našim modelima dali život možemo ih prikazati na internetu, u web pregledniku, bez ikakvih dodatnih programa te uz pomoć toga dižemo cijelo korisničko iskustvo na veću razinu.

Ključne riječi: 3D model, biblioteka, korisničko iskustvo, Three.js, web preglednik

Abstract

Creating 3D graphic for web page using the Three.js library

In this thesis, the topic of which is the use of the Three.js library for displaying 3D content in web browsers, we learned what is needed to create the 3D models themselves and how to present them on the web.

Also, which programs should we use and what should we pay attention to when creating them. In order to show the life of our models on the Internet, in a web browser, without any additional programs, and with the help of this, we raise the entire user experience to a higher level.

Keywords: 3D model, library, Three.js, user experience, web browser

Životopis

Tomislav Slukan, student na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek. Rođen je u Osijeku 27.01.2000. godine. Svoje srednjoškolsko obrazovanje završio je također u Osijeku u Elektrotehničkoj i prometnoj školi te stekao zvanje Tehničar za računalstvo.

Prilozi

- Web adresa praktičnog djela rada: https://slukan-zavrsni.netlify.app/
- Adresa do GitLab repozitorija gdje se nalazi kod praktičnog djela rada: https://gitlab.com/tomislavslukan278/ferit-zavrsni-rad