Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

SANDI IVANUŠEC

OSTVARIVANJE 3D GRAFIKE U INTERNET PREGLEDNICIMA POMOĆU BIBLIOTEKE THREE.JS

Diplomski rad

Pula, lipanj, 2021.

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

SANDI IVANUŠEC

OSTVARIVANJE 3D GRAFIKE U INTERNET PREGLEDNICIMA POMOĆU BIBLIOTEKE THREE.JS

Diplomski rad

JMBAG: 0303069339, redoviti student

Studijski smjer: Informatika

Predmet: Izrada informatičkih projekata

Mentor: doc. dr. sc. Nikola Tanković

Pula, lipanj, 2021.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani _____, kandidat za magistra _____ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, _____, _____ godine

IZJAVA

o korištenju autorskog djela

Ja, _____ dajem odobrenje Sveučilištu Jurja Dobrile
u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom _____

_____

_____ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.
Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, _____ (datum)

Potpis

_____

Sadržaj

1. Uvod u Three.js

1.1 Uvod

Za iscrtavanje jednog trokuta u tri dimenzije na web stranici pomoću WebGLa treba mnogo linija računalnog koda. Pojavom mnogih biblioteka za prikaz 3D objekata olakšao se proces kodiranja u WebGLu. Jedna od poznatijih takvih biblioteka je Three.js. Osim cjelovitog opisa i analize mogućnosti Three.jsa, izrađen je i projekt koji web prodaju čini zanimljivom i interaktivnom.

1.2 Usporedba WebGLa i Three.jsa

Prije nekoliko godina nije bilo moguće pronaći mnogo resursa vezanih uz WebGL i Three.js. Iz toga razloga, bilo ih je teško naučiti. Svi oni koji imaju iskustva s WebGLom ili OpenGLom svjesni su koliko teško je napraviti čak i one najjednostavnije stvari. Three.js biblioteka olakšava programeru da određene stvari napravi puno lakše i brže. Upotrebom Three.js biblioteke dobivamo veliku fleksibilnost tj. možemo se više usmjeriti na ono što želimo postići nego na pisanje samoga koda. No bez obzira na to, 3D programiranje i dalje može postati zahtjevno pri izradi složenijih projekata jer prolazi kroz više disciplina poput programiranja (Three.js, WebGL, JavaScript) i matematike.

1.2.1 WebGL

WebGL (engl. Web Graphics Library) je JavaScript API za prikazivanje interaktivne 2D i 3D grafike u bilo kojem kompatibilnom web pregledniku bez upotrebe dodataka. U potpunosti je integriran s ostalim web standardima, što omogućuje grafičkim karticama brže izvršavanje fizike, obrade slika i efekata. WebGL računalni kod izvršava se na grafičkim karticama. Za razliku od procesora koji rade brze kalkulacije jednu po jednu, grafičke kartice rade sporije ali zato mogu raditi na tisuću kalkulacija istovremeno.

Sve do pojave WebGLa, za prikaz 3D sadržaja u web preglednicima trebalo je koristiti dodatak (engl. plugin) gdje je jedan od najpoznatijih bio Shockwave. Današnji web developeri imaju veliki izbor alata za grafiku poput: CSSa, Canvasa, SVG i naravno WebGLa. Svaka od ovih tehnologija ima svoje prednosti i mane. Razlog radi kojega bi trebali koristiti WebGL je izrada grafičkih efekata, koji jednostavno nisu mogući ili ih je teško implementirati u nekim od drugih grafičkih tehnologija poput: Canvasa i SVGa. Za razliku od ostalih tehnologija poput Flasha i Silverlighta, WebGLu nije potreban plugin kako bi radio. Neovisnost o platformi odlična mu je strana kao i podržanost od strane svih modernih web preglednika uključujući i Internet Explorer. Tehnologije za izradu web stranica poput CSSa, SVGa i Canvasa možemo kombinirati s WebGLom. WebGL bio je razvijen s namjerom da bude što brži a ne sigurniji, iz toga razloga neki od programera i dalje smatra da postoje sigurnosne rupe unutar njega.

Početkom 2009. godine, neprofitna tehnološka grupa Khronos Group pokrenula je WebGL Working Group uz početno suradnju s velikim tvrtkama poput Applea, Googlea, Mozillae, Opere i drugih. U ožujku 2011. objavljena je WebGL verzija 1.0. Zanimljivo je da i popularni softveri za izradu videoigara poput Unreal Enginea i Unitya podržavaju WebGL. Kao i za bilo koji drugi grafički API, stvaranje 3D sadržaja za WebGL zahtjeva upotrebu uobičajenih alata namijenjenih za to i mogućnost izvoza 3D objekta u format koji neka od WebGL biblioteka može pročitati. U tu se svrhu mogu koristiti softveri za 3D dizajn poput Blendera, Autodesk Mayae ili SimLab Composera. Blend4Web jedan je od zanimljivih alata koji omogućuje izradu cijele WebGL scene u softveru Blender kao i izvoz u preglednik jednim klikom. Postoje i mrežne platforme poput Sketchfaba i Clara.io koji omogućuju korisnicima izravno prenašanje 3D modela, kao i njegovo prikazivanje pomoću WebGL preglednika koji se nalazi na tim web stranicama.

Ideja WebGLa je prikazivanje trokuta izvanrednom brzinom. Takvi trokuti mogu biti različitih oblika. Pri crtanju 3D modela, ideja je nacrtati više trokuta na određenim pozicijama. Grafička kartica će u tom slučaju pozicionirati sve točke odjedanput prema mnogim čimbenicima. Nakon što su sve točke pozicionirane, grafička kartica će nacrtati svaki piksel tih trokuta. Instrukcije koje govore gdje postaviti točke i nacrtati piksele su zapisane u tkz. shadersima. Shaderima pružamo hrpu informacija kao što su položaj točaka, transformacije modela, koordinate kamere i sl. Pojavom jednostavnijih 3D biblioteka ne treba zanemariti izvorni WebGL zato što je on i dalje koristan. WebGL postoji na niskoj razini što omogućuje optimizacije i veću kontrolu.

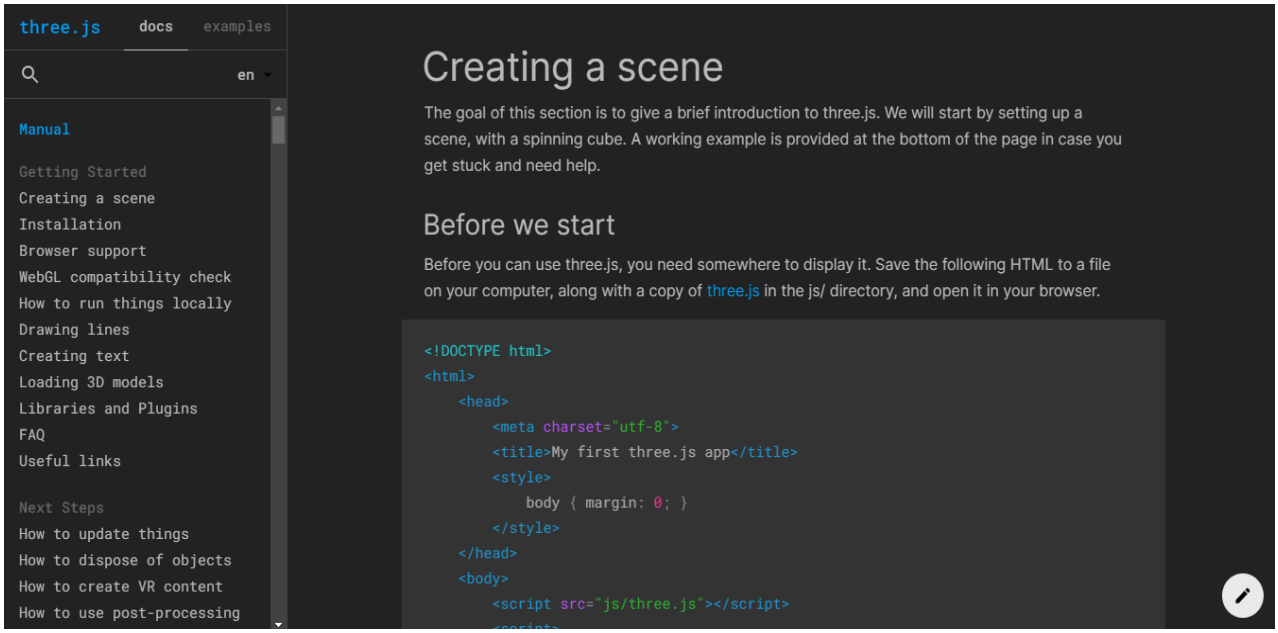
WebGL frameworci omogućuju brzo i jednostavno stvaranje WebGL sadržaja. Neki od najpoznatijih frameworka su:

- **A-Frame** - Web framework za izgradnju 3D, Augmentet Reality i Virtual Reality iskustava.
- **Away3D** - TypeScript / JavaScript adaptacija Away3D enginea izgrađena u Flashu.
- **Babylon.js** - JavaScript framework za izradu 3D igara s HTMLom 5 i WebGLom.
- **Clara.io** - Web softver za 3D računalnu grafiku razvijen od strane softverske tvrtke Exocortex.
- **CopperLicht** – JavaScript biblioteka i API otvorenog koda za stvaranje igara i interaktivnih 3D aplikacija pomoću WebGLa.
- **JanusWeb** - WebVR klijent otvorenog koda za zajedničku izgradnju i istraživanje 3D svijeta.
- **Kubity** - Kubity je mrežna platforma koja nudi različite načine prikazivanja, istraživanja i dijeljenja 3D modela u web pregledniku
  i mobilnim uređajima.
- **LayaAir** - API otvorenog koda za igre i module multimedijskih rutina. Omogućuje prikaz animacija na web pregledniku i mobilnim uređajima.
- **OSG.JS** - WebGL framework otvorenog koda zasnovan na konceptima OpenSceneGrapha.
- **Play Canvas** - 3D game engine otvorenog koda uz platformu za izradu hostinga u oblaku koja omogućava uređivanje unutar web preglednika.
- **Sketchfab** - Web stranica koja se koristi za prikaz i dijeljenje 3D sadržaja na Internetu.
- **Three.js** - JavaScript biblioteka koja se koristi za stvaranje i prikaz animirane 3D računalne grafike u web pregledniku.
- **Unity** - Unity je 2D i 3D game engine za razvoj igara na više platformi.

- **Verge3D** - WebGL framework prilagođen umjetnicima, integriran s 3DS Maxom i Blenderom, s PBR Shaderima, vizualnim programiranjem i mogućnošću izvoza za Facebook.
- **Wonderland Engine** - Razvojna platforma usmjerena na WebXR.

1.2.2 Three.js

Three.js je JavaScript 3D biblioteka koja se koristi za stvaranje i prikaz animirane 3D računalne grafike u web pregledniku pomoću WebGLa. Stvaranje JavaScript 3D biblioteke opće namjene koja će biti jednostavna za korištenje, lagana u smislu performansi i koja će omogućiti rad na različitim web preglednicima, bio je primarni cilj ove biblioteke. Three.js radi u svim preglednicima koji podržavaju WebGL 1.0. U samim početcima pokazao se kao jako dobra biblioteka, međutim dokumentacija nije bila idealna. S vremenom je počeo biti sve veći, pa je samim time bilo i sve teže održavati njegovu dokumentaciju. No s vremenom stvari su se polako ispravile te je današnja dokumentacija Three.jsa i više nego dobra. Dokumentacija prikazuje puno raznovrsnih i specifičnih primjera računalnog koda, te je moguće u relativno malom vremenu pronaći stvari koje tražimo.
Bez obzira što se ona naglo razvija, biblioteka je trenutno stabilna.
Ona je zaista velika i treba mnogo vremena kako bi se Three.js naučio za izradu ozbiljnijih projekata. Three.js radi s WebGLom, no može raditi i sa SVGom i CSSom, ali moramo imati na umu da su jako ograničeni.
Korištenje SVGa i CSSa neće dati najbolje rezultate i performanse, pa se za kompleksnije projekte preporučuje korištenje Three.jsa. On nam omogućuje drastično pojednostavljenje procesa izradom raznih 3D modela koje bi bilo teško izraditi izvornim WebGLom. Mogućnost interakcije s WebGLom omogućuje nam vlastitu izradu shadera i unosa podataka. Računalni kod Three.jsa prvi je put objavio Ricardo Cabello na GitHubu u travnju 2010. Računalni kod je u početku bio razvijen u ActionScriptu, 2009. godine prenesen je na JavaScript. Dva velika razloga radi kojih je Ricardo Cabello prešao na JavaScript su: neovisnost platforme i to što se računalni kod nije morao kompajlirati prije svakog izvođenja. Three.js je znatno pomogao programerima u razvoju web stranica i konstantno se poboljšava. Zanimljivo je da je za učenje Three.jsa dovoljno koristiti samo web preglednik, no radi jednostavnosti se preporuča korištenje i drugih alata.



Slika 1 - Prikaz odlične Three.js dokumentacije koja omogućuje brz pronalazak onoga što tražimo i to uz puno primjera

Three.js uključuje sljedeće značajke:

- Efekti
- Scene
- Kamere
- Animacije
- Svjetla
- Materijali
- Sjenčanja
- Objekti
- Geometrije
- Učitavači podataka
- `Utilities` (skup vremena i 3D matematičkih funkcija)
- Izvoz i uvoz
- Podrška

- Primjeri
- Otklanjanje pogrešaka
- Virtualna i proširena stvarnost

1.3 Načini pokretanja osnovne Three.js aplikacije

Postoji više načina kako u projekt uključiti Three.js biblioteku. Ovdje će biti opisana tri različita načina uključivanja:

1) Putem URL poveznice
2) Putem kopije Three.jsa u js datoteku
3) Instalacija putem NPMa

1.3.1 Putem URL poveznice

Prvo trebamo otvoriti IDE tj. okruženje u kojem želimo pisati kod (npr. `Visual Studio Code`). Možemo kreirati ili otvoriti već izrađeni folder u kojem ćemo spremati sve dijelove našega projekta. Izraditi ćemo novu datoteku (engl. `file`) koju ćemo nazvati **index.html.** U **index.html** napisati ćemo dolje priloženi kod koji predstavlja najosnovniji `hello world` primjer three.js aplikacije:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Moja prva three.js aplikacija</title>
    </head>
    <body>
```

```
        <script src="https://threejs.org/build/three.js"></script>
        <script>
            const scene = new THREE.Scene();
            const camera = new THREE.PerspectiveCamera( 75, window.innerWidth / wi
ndow.innerHeight, 0.1, 1000 );

            const renderer = new THREE.WebGLRenderer();
            renderer.setSize( window.innerWidth, window.innerHeight );
            document.body.appendChild( renderer.domElement );

            const geometry = new THREE.BoxGeometry();
            const material = new THREE.MeshBasicMaterial( { color: 0x03fcf4 } );
            const cube = new THREE.Mesh( geometry, material );
            scene.add( cube );

            camera.position.z = 5;

            const animate = function () {
                requestAnimationFrame( animate );

                cube.rotation.x += 0.01;
                cube.rotation.y += 0.01;

                renderer.render( scene, camera );
            };

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

S pomoću ove linije koda smo u gornje prikazanom kodu uključili Three.js biblioteku u svoj projekt koristeći URL poveznicu:

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

Kako bismo pokrenuli aplikaciju, kliknemo desnim klikom miša unutar mjesta gdje smo napisali programski kod (.html datoteka) i zatim odaberemo `open with Live Server`.

U slučaju da nemamo tu opciju, možemo preuzeti `Live Server` ekstenziju za `Visual Studio Code` pod izborom `Extensions`.

1.3.2 Putem kopije Three.jsa u js datoteku

U `Visual Studio Codeu` možemo navigirati do već izrađene mape za projekt ili izradimo novu. Unutar te iste mape izradimo folder pod nazivom **js**. Unutar mape **js** izradimo datoteku sa nazivom **three.js**. Otvorimo web preglednik i navigiramo na URL adresu: **threejs.org/build/three.js** Pritisnemo dvije tipke istovremeno **CTRL + A** kako bismo označili cijeli kod, te **CTRL + C** kako bismo cijeli kod kopirali. U `visual Studio Codeu` zalijepimo kopirani kod u **three.js** datoteku koju smo prethodno kreirali. Unutar `body html taga` napišemo ovu liniju koda kako bismo uključili three.js biblioteku u naš projekt:

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

1.3.3 Instalacija putem NPMa

U `Visual Studio Codeu` možemo navigirati u praznu mapu našega projekta koju smo prije stvorili. Odaberemo **Terminal** -> **New Terminal** kako bi nam se otvorio novi terminal. Unutar terminala napišemo **npm init** te prođemo kroz sve korake. Ovom komandom stvorila se datoteka **package.json**. Kako bismo preuzeli biblioteku Three.js unesemo komandu **npm install --save three**. Radi lakšeg navigiranja možemo otvoriti **node_modules** mapu, kliknuti i držati lijevim klikom te povući mapu **build** izvan svega tj. na `root` mjesto. Stvorimo praznu .html datoteku s nazivom **index.html** i unutar nje napišemo osnovni kod kojeg smo gore napisali. Iz nepoznatog razloga (bar je tako u mom slučaju), treba minimizirati `Visual Studio Code` te otvoriti datoteku projekta i navigirati prema **node_modules** > **three** > **build** te 'izrezati' mapu `build` i 'zalijepiti' ju na početak izvan svih mapa tj. u `root` mapu. Vrateći se u `Visual Studio Code`, možemo primijetiti automatsko ažuriranje. Nakon toga, u html tag `<body>` dovoljno je napisati ovu liniju koda:

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

2. Osnovni elementi Three.jsa

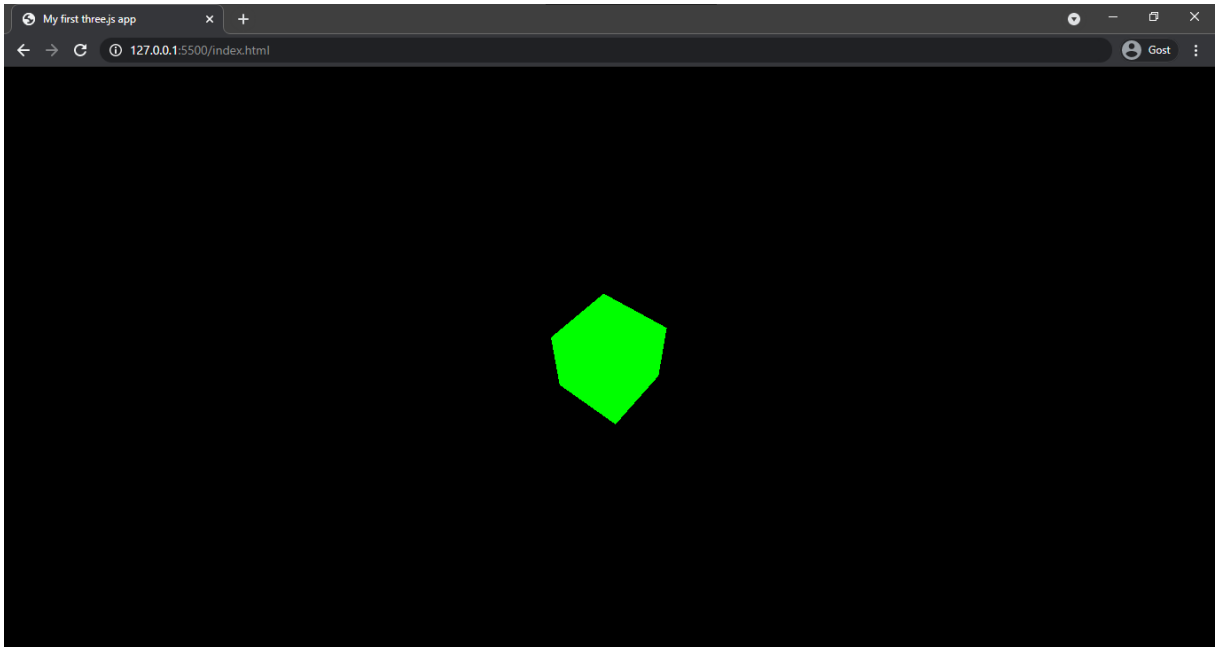
2.1 Jednostavna 3D scena

Za kreiranje jednostavne scene dovoljno je imati dvije datoteke: jednu JavaScript datoteku i jednu HTML datoteku. Možemo imati i samo jednu HTML datoteku unutar koje osim HTML koda pišemo i JavaScript kod. Preporučuje se da se ovakve datoteke rade unutar `code editora` a ne unutra `file explorera` kako bismo izbjegli greške i vidjeti skrivene datoteke.
Ako u kodu imamo grešku, možemo ju detaljnije proučiti u konzoli web preglednika. U mapi `build` postoje 3 vrste Three.js biblioteke: three.js (cijela Three.js biblioteka), three.min.js (cijela Three.js biblioteka ali smanjena po veličini tj. kompresirana), three.module.js (koristi se za upotrebu modula).

Minimalno 4 elemenata potrebno je kako bismo napravili scenu:

- Scena
- Objekt
- Kamera
- `Renderer`

Učitavanje Three.js biblioteke pomoću `<script>` ima limitacije tj. ne uključuje neke od klasa. Jedan od razloga je veličina a drugi je radi sigurnosnih razloga.



Slika 2 – Prikaz rotirajuće zelene kocke koja predstavlja jednostavni `Hello World` Three.js biblioteke

2.1.1 Bundler

`Bundler` je alat kojem šaljemo JavaScript, CSS, HTML, slike, TypeScript, Stylus, Sass računalni kod. Drugim riječima, `Bundler` je alat koji spaja naš računalni kod i sve njegove ovisnosti u jednu JavaScript datoteku. Danas ih ima mnogo, najpopularniji su: `Browserify` i `Webpack`

2.1.2 Webpack

Pozitivne strane Webpack-a su: njegova popularnost, odlična dokumentacija i dobro održavanje. Jedna i možda jedina loša strana Webpack-a je ta što ga je teško konfigurirati. Shvatiti ga, dosta je teško ali zato pruža jako puno mogućnosti.

2.2 Geometrije

Geometrija je sastavljena od vrhova (koordinatne točke u 3D prostoru) i lica (trokuti koji spajaju vrhove kako bi stvorili površinu). Vrhovi mogu sadržavati različite stvari poput: pozicije, UV koordinate, boje, nasumične vrijednosti, veličine. Postoji više različitih geometrija poput:

- `BoxGeometry`
- `PlaneGeometry`
- `CircleGeometry`
- `ConeGeometry`
- `CylinderGeometry`
- `RingGeometry`
- `TorusGeometry`
- `TorusKnotGeometry i DodecahedronGeometry`
- `OctahedronGeometry`
- `TetrahedronGeometry`
- `IcosahedronGeometry`
- `SphereGeometry`
- `ShapeGeometry`
- `TubeGeometry`
- `ExtrudeGeometry`
- `LatheGeometry`
- `TextGeometry`

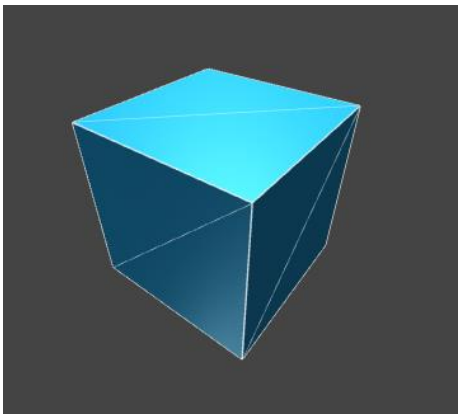
Kombinacijom svih ovih geometrija možemo stvoriti kompleksne oblike. Stvaranje kompleksnih 3D oblika je zahtjevno, pa se preporuča korištenje raznih softvera za 3D modeliranje.

2.2.1 BoxGeometry

Ova geometrija ima 6 parametara: `width`, `height`, `depth`, `widthSegments`, `heightSegments` i `depthSegments`.

Zadnja 3 označavaju koliko je geometrija podijeljena po svakoj od osi tj. od koliko trokuta se treba sastojati lice geometrije. Ako želimo imati uvid u trokute možemo upotrijebiti **wireframe: true** na materijalu.
Zanimljivo je da možemo stvoriti vlastitu geometriju. To možemo učiniti s:

```
const geometry = new THREE.Geometry()
```



Slika 3 – Prikaz geometrije: BoxGeometry

2.2.2 Buffer geometrije

`Buffer` geometrije su učinkovitije i optimiziranije, ali manje pogodne za programere. Grafička kartica računala ovakve će geometrije izvoditi puno lakše.

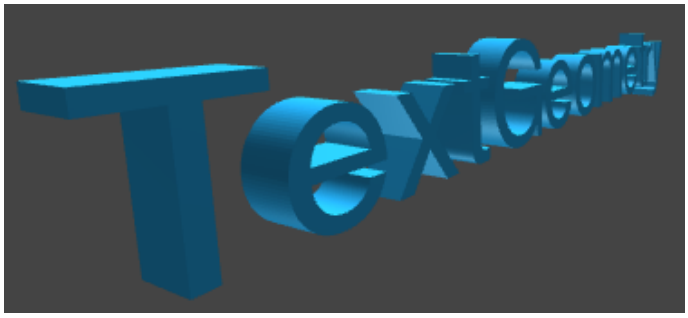
2.2.3 Indeks

Neke od geometrija imaju lica koja dijele zajedničke vrhove. Vrhove i indekse za stvaranje lica možemo odrediti prilikom stvaranja `Buffer` geometrije. Ovakve vrhove možemo upotrijebiti više puta. Ovakav način omogućuje slanje manjeg broja podataka grafičkoj kartici, no za ovako nešto potrebno je više truda oko organizacije koda.

2.2.4 3D tekst

Klasa `TextBufferGeometry` koristi se za stvaranje 3D teksta. Ako već posjedujemo font koji nam se sviđa, možemo ga konvertirati s online alatom poput **Facetype.js**a. Naravno, uvijek možemo koristiti i fontove koji nam dolaze uz Three.js.

Za učitavanje fonta koristimo **FontLoader**. Preporuka je ne koristiti tekst više puta u projektu zbog opterećivanja uređaja na kojemu se izvodi. Kako nam se pri rotaciji tekst ne bi okretao oko prvog slova, dobro ga je centrirati u sredinu. Jedan od načina centriranja teksta je koristeći tkz. `Bounding` koji predstavlja informacije povezane s geometrijom. To može biti npr. kocka ili kugla. Three.js ovaj način koristi kako bi `renderirao` samo one objekte koji su unutar kocke ili kugle, kako bi performanse na stranici bile što bolje. Drugi način za centriranje teksta je koristeći **center()** metodu.



Slika 4 – Prikaz 3D tekst geometrije

2.3 Materijali

Za postavljanje boja na geometriju koriste se materijali. Algoritmi su zapisani u programima koji se nazivaju `shaders`. Ne moramo programski pisati `shadere`, već možemo koristiti ugrađene materijale. Najosnovniji materijal bez boje je `MeshBasicMaterial`, njega možemo instancirati na sljedeći način:

```
const material = new THREE.MeshBasicMaterial();
```

Ako imamo više 3D objekata, performanse će nam biti bolje ukoliko na svima primijenimo isti materijal. Kod većine materijala svojstva se mogu postaviti na dva načina:

```
// Direktno kod instanciranja
const material = new THREE.MeshBasicMaterial({
     map: doorColorTexture
})
// Nakon instanciranja
const material = new THREE.MeshBasicMaterial()
material.map = doorColorTexture
```

Interesantno je to što možemo kombinirati `map` i `color`, dakle teksturu i boju zajedno.

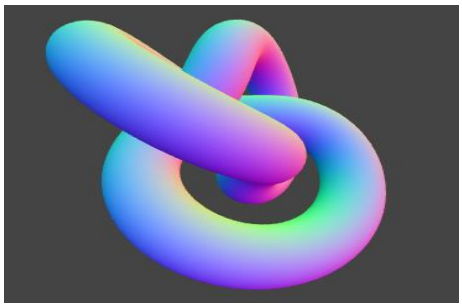
- `Wireframeom` možemo prikazati trokute od kojih su sastavljeni 3D objekti tj. geometrija 3D objekata.
- `Opacityom` možemo kontrolirati neprozirnost 3D objekata, no moramo imati na umu da za ovu opciju postavimo **transparent = true**.
- `AlphaMapom` kontroliramo prozirnost pomoću teksture. Možemo ovo zamisliti kao pravokutnu sliku koja ima unutrašnju boju bijelu a vanjsku crnu. Ono što je bijelo biti će vidljivo, dok će ono što je crno biti sakriveno. Naravno, i ovdje moramo imati na umu da postavimo **transparent = true**.
- `Side` nam omogućuje da odlučimo koja je strana 'lica' vidljiva. Imamo tri mogućnosti: `FrontSide`, `BackSide`, `DoubleSide`.

2.3.1 Mesh Normal Material

Ovako možemo alocirati mjesto u memoriji za ovaj materijal:

```
const material = new THREE.MeshNormalMaterial();
```

Ovaj materijal prikazuje ljubičastu boju. Riječ `Normal` označava podatke koji sadrže smjer vanjske strane lica. `Normals` se mogu koristiti za svjetla, refleksije i refrakcije. `MeshNormalMaterial` dijeli slična svojstva kao i `MeshBasicMaterial` poput: `wireframe`, `transparent`, `opacity`, `side`, no tu je i dodatno svojstvo `flatShading`. Najčešće korišten materijal za otklanjanja pogreški `normalsa je MeshNormalMaterial`.



Slika 5 – Prikaz geometrije: MeshNormal

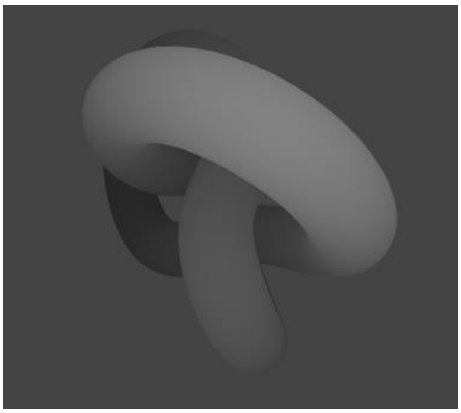
2.3.2 Mesh Matcap Material

`MeshMatcapMaterial` prikazati će boju koristeći `normals` kao referencu kako bi uzeo pravu boju za teksturu koja izgleda kao kugla. Kada koristimo slike kao teksture, `MeshMatcapMaterial` će uzeti u obzir boje unutar slike i postaviti ih kao boje na 3D objekte. Na ovaj se način može simulirati svjetlo bez uporabe svjetla u sceni.
Jako puno `Matcaps` tekstura može se pronaći besplatno na internetu, no ako radimo web stranicu za klijenta bilo bi dobro zatražiti odobrenje od vlasnika `matcapa`. Druga opcija je izraditi vlastite `matcapove` koristeći 3D ili 2D softvere.

Slika 6 – Prikaz geometrije: MeshMatcap

2.3.3 Mesh Depth Material

`MeshDepthMaterial` promijeniti će boju geometrije u bijelu ako kamera dođe blizu ili u crnu ako se kamera udalji od 3D objekta. Na ovaj način može se dobiti bolji dojam dubine.

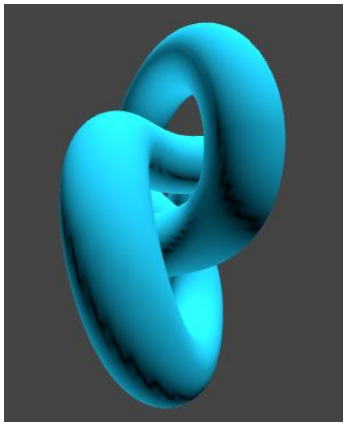


Slika 7 – Prikaz geometrije: MeshDepth

Sljedećim navedenim materijalima biti će potrebno svjetlo.

2.3.4 Mesh Lambert Material

`MeshLambertMaterial` je materijal koji reagira na svjetlo. Ovaj materijal ima nova svojstva vezana uz svjetla. Ovaj materijal je dobar što se tiče performansi ali se na geometriji mogu primijetiti neobični uzorci.

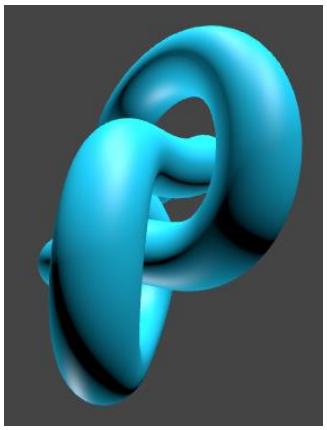


Slika 8 – Prikaz geometrije: MeshLambert

2.3.5 Mesh Phong Material

Jako sličan `MeshLambertMaterialu` je `MeshPhongMaterial`, no ovdje nemamo više neobičnih uzoraka na geometriji.

Osim toga, može se vidjeti i svjetlosna refleksija koju možemo kontrolirati pomoću `shininess`. Omogućena nam je i kontrola nad bojom refleksije pomoću `specular`.



Slika 9 – Prikaz geometrije: MeshPhong

2.3.6 Mesh Toon Material

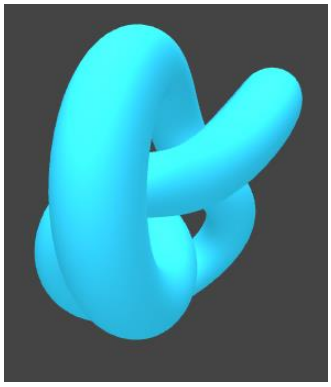
`MeshToonMaterial` je sličan `MeshLambertMaterialu`, no ovaj nalikuje više materijalima koji se koriste npr. u crtanim filmovima. Za mogućnost mijenjanja više svojstava boja može se koristiti `gradientMap` i `gradientTexture`.

Slika 10 – Prikaz geometrije: MeshToon

2.3.7 Mesh Standard Material

Najbolji materijal od svih koji koristi `PBR` je `MeshStandardMaterial`. `PBR` predstavlja principe koji pokušavaju oponašati uvjete stvarnog života.

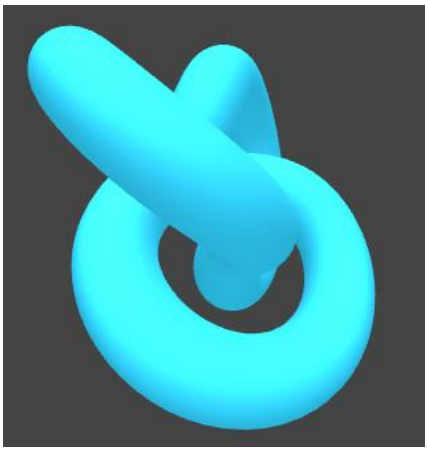
Poput `MeshLambertMateriala` i `MeshPhongMateriala` podržava svjetla ali s realističnijim algoritmom i boljim parametrima poput `roughness` i `metalness`.



Slika 11 – Prikaz geometrije: MeshStandard

2.3.8 Mesh Physical Material

`MeshPhysicalMaterial` jako je sličan materijalu `MeshStandardMaterial`. Za razliku od `MeshStandardMateriala`, `MeshPhysicalMaterial` ima podršku efekta prozirnog sloja.



Slika 12 – Prikaz geometrije: MeshPhysical

2.3.9 Points Material

`PointsMaterial` možemo koristiti kako bismo napravili tkz. čestice.

2.3.10 Shader Material i Raw Shader Material

`ShaderMaterial` i `RawShaderMaterial` oboje se mogu koristiti za stvaranje vlastitih materijala.

2.3.11 Environment Map

`Environment map` je slika koja okružuje scenu. Može se koristiti za refleksiju ili lom svjetla, ali i za opće svjetlo. Three.js podržava samo `environment mape` u obliku kocke. Kako bismo učitali teksturu u obliku kocke, moramo koristiti `CubeTextureLoader` umjesto dosadašnjeg `TextureLoadera`.

Primjere `Environment mapa` možemo pronaći u mapama Three.jsa putanjom: **/static/textures/environmentMap/**. Osim toga možemo ih pronaći i na internetu, no moramo provjeriti imamo li pravo na njihovo korištenje. Jedna od dobrih web stranica za besplatne mape koje možemo koristiti i u komercijalne svrhe je: **https://hdrihaven.com/**. Mape na ovoj stranici su u HDR formatu, no lako ih možemo raznim `online converterima` pretvoriti u mapu u oblik kocke.

Slika 13 – Prikaz EnvironmentMape unutar web stranice

2.4 Teksture

Slike koje će prekriti površinu geometrija zovu se teksture. Postoji više tipova tekstura s više različitih efekata.

Neke od tekstura su:

- `Color` - najjednostavnija tekstura, primjenjuje se na geometriju
- `Alpha` - slika u sivim tonovima, bijeli dio slike biti će vidljiv dok crni neće
- `Height` - slika u sivim tonovima, ako je boja bijela, vrhovi će ići gore a ako je crna, spuštati će se dolje
- `Normal` – slika u plavim tonovima, mogu se dodati detalji, ima bolje performanse za razliku od `Height` teksture
- `Ambient occlusion` - slika u sivim tonovima, dodaje lažne sjene u pukotine, nije fizički točan, pomaže stvoriti kontrast i vidljivost detalja
- `Metalness` - slika u sivim tonovima, bijela je metalna, crna je nemetalna, uglavnom se koristi za refleksiju
- `Roughness` - slika u sivim tonovima, bijela je gruba dok je crna glatka, uglavnom se koristi kada se želi postići rasipanje svjetlosti

Postoji još mnogo tekstura, no gore navedene su osnovne koje se koriste u praksi.

Navedene teksture (posebno `metalness` i `roughness`) slijede tkz. `PBR` principe. `PBR` je kratica za `Physically Based Rendering`. Ideja `PBRa` su primjena više tehnika koje teže slijediti upute iz stvarnog života kako bi postigle realne rezultate.

2.4.1 Učitavanje tekstura

Sliku ne možemo koristiti direktno, već ju moramo pretvoriti u teksturu. Najlakši način za učitavanje teksture je pomoću `TextureLoadera` s dvije linije koda:

```
const textureLoader = new THREE.TextureLoader()
const texture = textureLoader.load('/textures/water/water.jpg')
```

Za učitavanje više tekstura dovoljan je jedan `TextureLoader`.

2.4.2 Loading Manager

Možemo upotrijebiti tkz. `LoadingManager` za mutalizaciju događaja. Ako želimo znati globalni napredak učitavanja ili biti informirani kada će se sve učitati, `LoadingManager` je onda jako koristan.

2.4.3 UV Unwrapping

`UV Unwrapping` možemo usporediti s razmotavanjem origamija ili omota od slatkiša. UV koordinate su generirane od strane Three.jsa, no ako želimo raditi sa svojom geometrijom, morat ćemo specificirati UV koordinate. Ako izrađujemo geometriju s nekim od 3D softvera, morati ćemo napraviti UV unwrapping.

2.4.4 Transformiranje teksture

Na geometriji postoji mogućnost ponavljanja teksture. Teksturu možemo i rotirati. Ako ju želimo rotirati za 180 stupnjeva koristiti ćemo **PI**, **PI/4** ako želimo zarotirati za 45 stupnjeva a PI * 2 ako želimo zarotirati za 360 stupnjeva.

2.4.5 Filtering i Mipmapping

Gledajući gornje lice npr. kocke (dok je to lice gotovo skriveno), primijetiti ćemo mutnu teksturu, što je posljedica filtriranja i mapiranja.

`Mipmapping` je tehnika koja se sastoji od kreiranja upola manje verzije teksture više puta sve dok ne dobijemo 1x1 teksturu. Sve varijacije tekstura poslane su grafičkoj kartici koja odabire najprikladniju verziju teksture.
Sve ovo riješeno je od strane Three.jsa i grafičke kartice, no imamo i mogućnost izabrati različite algoritme. Postoje dva tipa filter algoritma:

- **`Minification filter`** - događa se kada su pikseli teksture manji od piksela `rendera`, drugim riječima, za površinu koju pokriva, tekstura je prevelika**.** Možemo promijeniti `Minification filter` teksture koristeći **minFilter**.
- **`Magnification filter`** - događa se kada su pikseli teksture veći od piksela `rendera`, drugim riječima, za površinu koju pokriva, tekstura je premala**.**

2.4.6 Format teksture i optimizacija

Kada pripremamo svoje teksture moramo imati na umu 3 jako bitna elementa:

- težinu teksture
- težinu slike (razlučivost)
- podatke koje stavljamo u teksturu

2.4.6.1 Težina teksture

Moramo imati na umu da će korisnici pri posjeti naše stranice na neki način 'morati skinuti teksture', u tom slučaju moramo uzeti u obzir i one korisnike koji imaju jako lošu internetsku vezu jer će se takvim korisnicima sporije učitavati web stranica.

Najčešće se preporučuje korištenje sljedeća dva formata za slike:

- .jpg - kompresija s gubicima, ali obično lakša po veličini
- .png - kompresija bez gubitaka, ali obično teža po veličini

Možemo koristiti i web stranice za kompresiju poput: **`TinyPNG`**

2.4.6.2 Težina slike

Svaki piksel teksture biti će pohranjen na grafičku karticu bez obzira na težinu slike. Moramo imati na umu da grafičke kartice imaju ograničeno mjesto za pohranu. Preporuka je pokušati smanjiti veličinu slike što je više moguće.

2.4.6.3 Podaci

Teksture podržavaju prozirnost, no prozirnost ne možemo imati u formatu **.jpg**. U tom nam je slučaju bolje koristiti format **.png**.

2.4.7 Gdje se mogu pronaći teksture ?

Odlične teksture, teško je pronaći. Neke od boljih web stranica gdje se mogu pronaći teksture su:

- **poliigon.com**
- **3dtextures.me**
- **arroway-textures.ch**

Naravno, uvijek nam preostaje i mogućnost izrade vlastite teksture sa slikama i 2D softverima poput npr. `Photoshopa`.

2.5 Svjetla

Dodavanje svjetlosti je jednostavno, instanciramo ga s pravom klasom te ga dodamo sceni. Svjetla sceni dodaju dubinu i realizam. Također se mogu koristiti za stvaranje specijalnih efekata i za realistični izgled materijala. Ako smo kao materijal koristili `MeshStandardMaterial`, njemu će obavezno biti potrebno svjetlo. Svjetla u Three.jsu usko su povezana s materijalima.

2.5.1 Ambient Light

'Najlakše' svjetlo u Three.jsu je `AmbientLight`. Ovo svjetlo ne može imati sjenu. Kao primjer za ovo svjetlo možemo zamisliti svjetlo u jako osvjetljenim uredima.
Kako bismo instancirali `AmbientLight` koristimo `AmbientLight` klasu:

```
const light = new THREE.AmbientLight( 0x404040, 0.4 );
scene.add( light );
```

Prvi parametar označava boju a drugi intenzitet. Intenzitet može imati vrijednost od 0 do 1. Parametre boje i intenziteta možemo pisati unutar zagrada ili zasebno. Ideja Ambient Lighta je imati direktnu svjetlost koja dolazi sa svih strana. Svaka strana objekta biti će osvjetljena na isti način bez obzira na njegov oblik. Postoji i naziv `Light Bouncing` koji označava odbijanje svjetlosti od tla nazad na objekt. No `Light Bouncing` se dosta teško izvodi u Three.jsu, pa ono što možemo učiniti je simulirati ga prigušenim svjetlom. Ovo svjetlo može biti nježnije inicijalizacijom sive boje.

2.5.2 Directional Light

Ovako možemo alocirati mjesto u memoriji za `Directional Light`:

```
const directionalLight = new THREE.DirectionalLight(0x00ffc,1);
scene.add(directionalLight);
```

Svjetlo koje dolazi paralelno u istom smjeru i koje možemo usporediti sa Suncem je `Directional Light`. Ovo svjetlo se u praksi jako puno koristi. Možemo ga isključiti ili upaliti postavljajući varijablu na `false` ili `true`.
Na početku je uvijek zadana bijela boja i intenzitet vrijednosti 1 koja je ujedno i maksimalna vrijednost dok je 0 najmanja moguća vrijednost. Na objektima će na nekim mjestima svjetlo biti vidljivo dok na drugima neće. Ako želimo promijeniti poziciju svjetla, možemo to učiniti pomoću ove linije koda:

```
directionalLight.position.set(0.25, 1, 0);
```

Ovo svjetlo, bez obzira na svoju poziciju, pokazivati će uvijek prema centru podloge. S obzirom na poziciju, neće se osjetiti velika razlika u promjeni svjetla ako ne koristimo sjene.

2.5.3 Hemisphere Light

Ovako možemo alocirati mjesto u memoriji za `Hemisphere Light`:

```
const hemisphereLight = new THREE.HemisphereLight(0xff0000, 0x0000ff, 0.5);
scene.add(hemisphereLight);
```

Možemo primijetiti da sada imamo 2 parametra kao boje. Kao prvu boju odabrali smo crvenu a kao drugu plavu. Crvena boja, s obzirom na podlogu, svijetlit će od gore prema dolje dok će plava boja svijetliti od dole prema gore. `Hemisphere Light` radi na sličan način kao i `Ambient Light` tj. dolazi sa svih strana. Između `Ambient Lighta` i `Hemishpere Lighta` najviše se preporučuje korištenje `Hemisphere Lighta` zato što nam omogućuje veću kontrolu nad svjetlom.

2.5.4 Point Light

Ovako možemo alocirati mjesto u memoriji za `Point Light`:

```
const pointLight = new THREE.PointLight(0xff9000, 0.2);
scene.add(pointLight);
```

Ovo svjetlo predstavlja malu svjetleću točku koja imitira svjetlost u svim smjerovima, možemo ju zamisliti kao žarulju. Kao i `DirectionalLight`, u praksi se jako puno koristi. Ovo svjetlo utječe samo na materijale poput: `MeshLambert` i `MeshPhong`. I ovdje možemo promijeniti poziciju svjetleće točke ako to poželimo:

```
pointLight.position.set(-1, 0.5, 1)
```

Postoji još parametara za `Point Light` a to su: `distance` i `decay`. Bez ovih dvoje parametara jačina svjetla na objektima bila bi ista bez obzira na udaljenost.
U početku je udaljenost zadana na 0 što označava udaljenost bez limita. S ova dva atributa moguće je kontrolirati udaljenost i propadanje svjetlosti. Glavni parametri poput `color` i `intensity su` opcionalni.

2.5.5 Rect Area Light

Ovako možemo alocirati mjesto u memoriji za `Rect Area Light`:

```
const rectAreaLight = new THREE.RectAreaLight(0x4e00ff, 2, 1, 1);
scene.add(rectAreaLight);
```

Ovo svjetlo najlakše možemo zamisliti kao pravokutnu studijsku rasvjetu za fotografiranje.

`Rect Area Light` svjetlo ima dva dodatna parametra koja označuju visinu i duljinu pravokutnog svjetla. `Rect Area Light` radi samo sa `MeshStandardMaterial`, `MeshPhysicalMaterial` i ne može imati sjene. `RectAreaLight` možemo pozicionirati sljedećom linijom koda:

```
rectAreaLight.position.set(- 1, 0, 1);
```

Orijentacija svjetla nije jednostavna, moramo ju izračunati. No ako radimo s 3D objektima onda je moguće upotrijebiti **lookAt()** metodu. Ovu metodu bitno je pozvati tek nakon što smo promijenili poziciju. Primjer linije koda za korištenje **lookAt()** metode:

```
rectAreaLight.lookAt(new THREE.Vector3())
```

Kada stvorimo `rectAreaLight` možemo mijenjati tkz. `roughness` i `metalness`. Mijenjajući ova dva parametra možemo stvoriti realističan dojam odbijanja svjetla o 3D objekte. Jedno od najkopleksnijih svjetla u Three.js biblioteci je `RectAreaLight`.
Ovo svjetlo mora se dodati u svjetlo a ne u scenu te se mora stvoriti posebna datoteka `RectAreaLightUniformsLib.js` u našemu projektu. Osim toga `RectAreaLight` trebalo bi stvoriti sa:

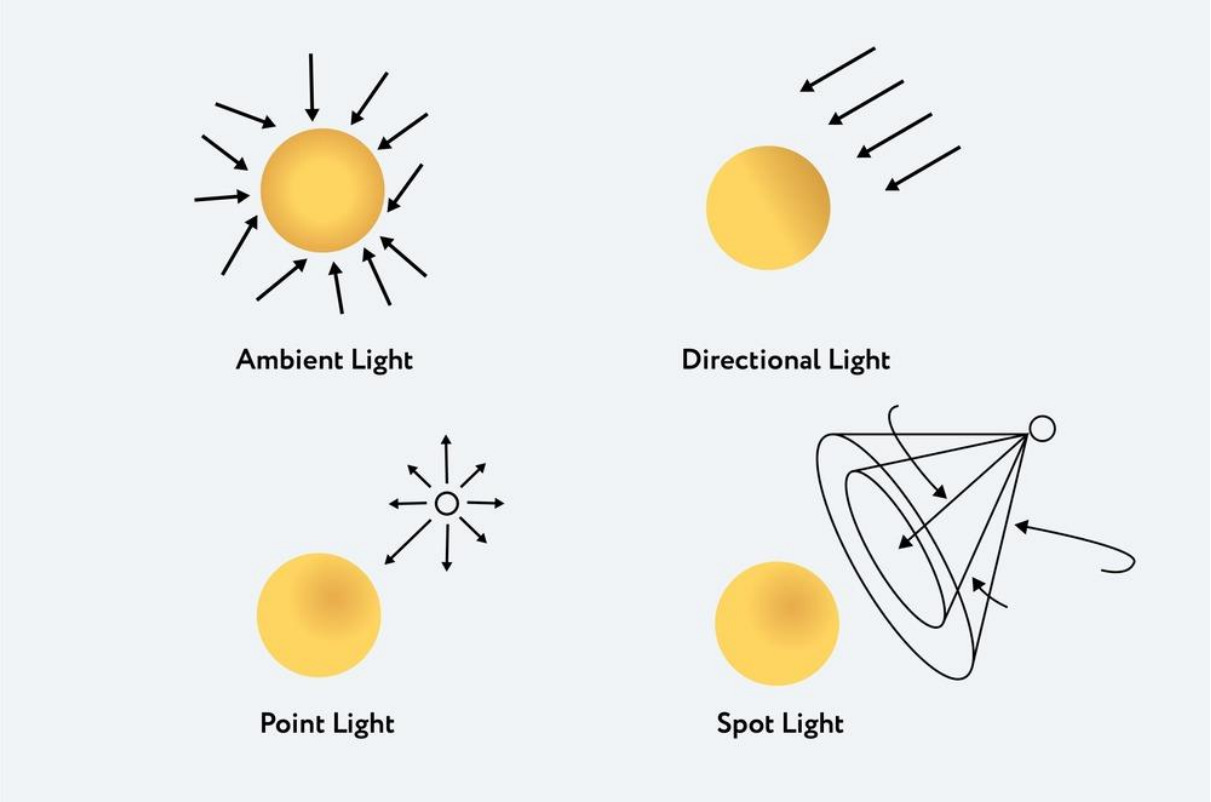
```
THREE.RectAreaLightUniformsLib.init()
```

2.5.6 Spot Light

Ovo svjetlo možemo zamisli kao baterijsku svjetiljku ili kao svjetlo u kazalištu. Svjetlo se emitira u jednome smjeru, što mu je veća udaljenost od podloge to će se više prostora osvijetliti. Za razliku od `DirectionalLighta`, `SpotLightom` možemo imati veću kontrolu nad svjetlom.
Ovako možemo alocirati mjesto u memoriji i postaviti poziciju za `Spot Light`:

```
const spotLight = new THREE.SporLight(0x78ff00, 0.3, 10, Math.PI * 0.2, 0.3, 1)
spotLight.position.set(1, 2, 3)
scene.add(spotLight)
```

Možemo primijetiti da imamo više parametara. Parametri su redom: boja, intenzitet, duljina, širina svjetlosti, oštrina na kraju svjetla, brzina kojom ide do limita.
I ovo svjetlo možemo pozicionirati ali ovaj puta moramo koristiti tkz. `target`. `SpotLight target` treba staviti u scenu, inače se neće ništa desiti. Kao i `RectAreaLight`, `SpotLight` je jedno od najkopleksnijih svjetla u Three.js biblioteci. `SpotLight` je najbolje koristiti u primjerima kao što su ulično svjetlo ili stolno svjetlo.



Slika 14 – Prikaz smjerova različitih vrsti svjetlosti

Na slici gore možemo vidjeti na koji način različite vrste svjetla osvjetljavaju objekte. `AmbientLight` aplicira svjetlo na sve objekte bez obzira na njihovu poziciju tj. oni se mogu nalaziti blizu ili jako daleko od kamere. `AmbientLight` je dobro primjenjivati kada trebamo obično svjetlo u sceni. `DirectionalLight` je svjetlo koje svijetli u jednom određenom smjeru. `PointLight` emitira svjetlo u svim smjerovima. `SpotLight` možemo najlakše zamisliti kao stožac ili svjetlo scenske produkcije.

2.5.7 Performanse

Korištenje puno svjetla odjedanput jako je loša ideja, ako to ipak želimo moramo biti svjesni da će se performanse smanjiti. Dobra je praksa koristiti što je manje svjetla moguće i svjetla koja su po performansama 'lakša'.
Svjetla koja nisu zahtjevna što se tiče performansi su npr. `AmbientLight` i `HemisphereLight`. Svjetla koja jako utječu na performanse su: `SpotLight` i `RectAreaLight`.

2.5.8 Baking

Ne preporučuje se korištenje Three.jsa ako trebamo koristiti puno svjetla. Ideja `Bakinga` je 'ispeći' svjetlo u teksture. Za to nije potrebno koristiti 3D softver, no u praksi se 3D softverima ipak postižu bolji rezultati. Nemogućnost kasnijeg pomicanja svjetla, jedan je od najvećih nedostataka `Bakinga`.
Za pomoć pri pozicioniranju svjetla možemo koristiti tkz. `helpere` koje možemo pronaći u dokumentaciji Three.js biblioteke. `Helperi` nam pomažu lakše vizualizirati cijelo svjetlo. Zbog nepoznatih razloga, može se desiti da nam neki od `helpera` prestanu raditi. Kako bismo riješili ovakve probleme nužno ih je ručno ažurirati. `Helpere` slobodno možemo ukloniti nakon što smo zadovoljni izrađenom scenom.

2.6 Sjene

Kako bi sve izgledalo realističnije, bilo bi dobro koristiti i sjene. Sjene koje inače možemo vidjeti na objektima zovu se `core shadows` dok prave sjene koje se pojavljuju iza nekog objekta zovu se `drop shadows`.

Sjene su oduvijek bile pravi izazov za programere koji su se morali snalaziti raznim trikovima kako bi se sjene prikazivale u realističnom `frame rateu` tkz. sličicama u sekundi. Three.js ima rješenje za to, no nije savršeno. Svjetlosni prikazi se pohranjuju kao teksture i nazivamo ih `shadow maps`. `Shadow map` je poput `renderirane` slike na kojoj se vide svi objekti u našoj sceni. S obzirom na `Shadow mape`, Three.js 'zna' gdje mora postaviti sjene.

2.6.1 Aktivacija sjene

Kako bi smo aktivirali sjene moramo provjeriti `renderer` tj. moramo mu 'reći' da želimo koristiti `Shadow mape`. To možemo učiniti ovako:

```
renderer.shadowMap.enabled = true
```

Postoje četiri vrste `Shadow mape`: `BasicShadowMap`, `PCF ShadowMap`, `PCFSoftShadowMap` i `VSMShadowMap`.
Sljedeći korak je proći kroz svaki objekt u sceni i odlučiti za svakoga da li će on stvarati sjenu pomoću `castShadow` i da li će ju primati pomoću `receiveShadow`.
Moramo imati na umu da samo određena svjetla omogućuju sjene, poput svjetla: `PointLight`, `DirectionalLight` i `SpotLight`.
Sjenu svjetla aktiviramo pomoću `castShadow` s ovom linijom koda:

```
directionalLight.castShadow = true
```

2.6.2 Optimizacija sjena

Sjene nam u nekim slučajevima neće izgledati savršeno, pa je potrebno provesti razne metode poboljšavanja sjena.
Što su manje vrijednosti, to će sjene biti preciznije, no ako su vrijednosti premale tada će sjene biti odsječene. Na `shadow mape` mogu se primijeniti različiti algoritmi koji će poboljšati performanse i izgladiti rubove.
Ako želimo pomiješati više sjena, zasigurno neće izgledati dobro te se oko rješavanja tog problema ne može ništa puno učiniti. Dobra ideja je postaviti širinu i visinu `mapSizea`.

Postavljanje kamere sjene što je bliže moguće dobro je za bolju kvalitetu sjene, preporučljivo je koristiti `camera helpere` kako bismo postavili idealne vrijednosti. `PointLights` i `SpotLights` koriste `PerspectiveCamera` za kameru sjene, prema tome, jedino je bitno dobro postaviti vrijednosti kamere 'blizu i daleko'. `DirectionalLights` koriste `OrtographicCamera`, prema tome vrijednosti poput: blizu, daleko, lijevo, desno, gore i dolje jako su bitne.

2.6.3 Physically Correct Lighting

Kao prvo za `renderer` moramo postaviti sljedeće:

```
renderer.physicallyCorrectLights = true
```

Što nam ukazuje na to, da će od sada pa na dalje biti omogućeno koristiti fizički ispravan način osvjetljenja.
Zatim moramo postaviti pravu vrijednost za `toneMappingExposure`. Isto tako moramo postaviti `gamma` ulaz i izlaz, `toneMapping` te svojstvo snage za svako svjetlo.

2.6.4 Baking sjene

`Baking` sjene su odlična alternativa za one koji žele imati puno sjena. Ovakve sjene integriramo u teksture koje nanosimo na materijale.
Prvi korak je uzeti teksturu koja može biti npr. obična slika. Drugi korak bio bi učitati tu teksturu putem tkz. `textureLoadera`.
Problem koji imamo kod `baking` sjena je taj da kada jednom učitamo teksturu na kojoj je 'zapečena' sjena, sama sjena ostaje na mjestu bez obzira što se npr. neki 3D objekt pomjerio na drugo mjesto. Postoji alternativa kojom možemo riješiti ovaj problem. Možemo učitati teksturu ispod npr. 3D objekta, pa ako pomaknemo 3D objekt, npr. prema gore zajedno s njim će se pomaknuti i tekstura. Osim pomicanja teksture zajedno s 3D objektom možemo smanjiti i intenzitet 'zapečene' sjene.

3. Napredni elementi Three.js-a

3.1 Uvozni modeli

Kako bismo stvorili kompleksne 3D modele bolje nam je koristiti vanjski 3D softver namijenjen upravo za to.
Možemo kompleksni model napraviti u 3D softveru, izvesti ga i zatim uvesti u Three.js. Postoji mnogo različitih formata modela iz različitih softvera gdje svaki format rješava određeni problem kao npr.

- podaci
- težina
- kompresija
- kompatibilnost
- autorska prava

Postoje različiti kriteriji formata poput:

- posvećen jednom softveru
- vrlo lagan, ali možda mu nedostaju specifični podaci
- gotovo svi podaci, ali su 'teški'
- `open source`
- nije `open source`
- binarni
- ASCII

Isto tako treba napomenuti da možemo kreirati i svoj vlastiti format. Neki od najpopularnijih formata su:

- OBJ
- FBX
- STL
- PLY
- COLLADA
- 3DS
- GLTF

Jedan od ovih formata počinje biti standard i trebao bi pokriti sve naše potrebe, a to je **GLTF**.

3.1.1 GLTF standardni format

GLTF je skraćenica od `GL Transmission Format`. Podržava različite skupove podataka poput: geometrija, materijala, kamera, svjetla, graf scene, animacije, kosture, morfiranje itd. On postaje standard u većini 3D softvera, razvojnih okruženja za igre, biblioteka i sl.
Prije nego što počnemo koristiti GLTF moramo se pitati bismo li i bez njega mogli nešto napraviti. Moramo se pitati kakvi podaci nam trebaju, koja je veličina datoteke, koliko nam je potrebno da dekompresiramo i sl.

GLTF je sam po sebi format, no možemo imati i različite GLTF formate poput:

- **gLTF** – standardni format, može sadržavati više datoteka u sebi
- **gLTF-Binary** – samo jedna datoteka, najčešće je 'lakša' što se tiče veličine, teško ju je promijeniti
- **gLTF-Draco** – sličnan gLTF standardnom formatu no podaci međuspremnika su zbijeni pomoću `Draco` algoritma, puno 'lakša' što se tiče veličine
- **gLTF-Embedded** – jedna datoteka poput gLTF-Binary formata, u JSON formatu, 'najteža' je što se tiče veličine

Odabiremo format u ovisnosti što trebamo postići, npr. ako želimo imati mogućnost mijenjati datoteke, možemo koristiti standardni gLTF format, no ako želimo imati samo jednu datoteku onda možemo koristiti gLTF-Binary format.

3.1.2 Učitavanje modela u Three.js

Prvo nam je potreban **GLTFLoader** kojeg možemo učitati iz `examples` mape pomoću ove linije koda:

```
import { GLTFLoader } from 'three/examples/jsm/loaders/GLTFLoader.js'
```

Te ga potom alociramo:

```
const GLTFLoader = new GLTFLoader()
```

Kao i kod ostalih `loadera`, uz **`GLTFLoader`** možemo koristiti i **`LoadingManager`**.
Za učitavanje modela u Three.js koristimo **load()** metodu koja ima 4 parametra:

- put do datoteke
- funkcija povratnog poziva
- funkcija povratnog poziva u tijeku
- funkcija povratnog poziva na pogrešku

3.1.3 Draco kompresija

`Draco` verzija može biti puno manja po veličini nego standardna verzija. Kompresija se primjenjuje na geometriju. Ako već koristimo standardnu verziju i na to nadodamo `Draco` verziju, ta verzija se neće koristiti tj. koristiti će se samo ako je potrebna.
Iako se čini da je najbolja opcija uvijek koristiti `Draco` radi kompresije, to nije uvijek slučaj. `Draco loader` je lakše koristiti ali ga je teže postaviti. Ima ga smisla koristiti najviše kada imamo modele velike veličine i kada je kompresija potrebna. Računalu je potrebno više vremena i resursa da dekodira kompresiranu datoteku. Dakle sve ovisi o situaciji i projektu.

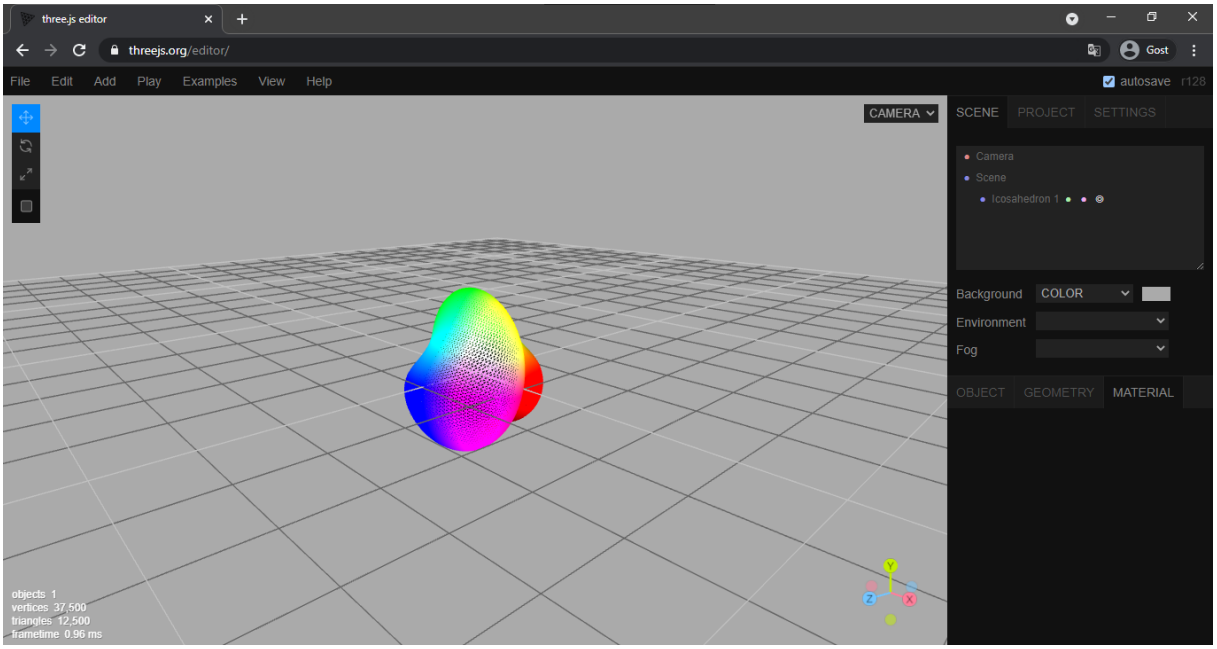
3.1.4 Animacije

U Three.jsu može se učitati i već animirani model. Izraditi animaciju izravno s Three.jsom nije lako. GLTF objekt sadrži u sebi svojstvo `animations`.
Moramo stvoriti tkz. `AnimationMixer`. `AnimationMixer` je poput 'igrača' povezanog s objektom koji može sadržavati jedan ili više `AnimationClipsa`.

3.1.5 Three.js editor

Three.js editor je mala 3D web aplikacija u koju možemo svoj preuzeti 3D model 'povući i ispustiti' unutar iste 3D web aplikacije.

Ovaj editor je koristan kada nam netko pošalje svoj 3D model kojeg trebamo istestirati, u tom slučaju ne moramo gubiti vrijeme učitavanjem tog istog modela unutar Three.jsa već ga možemo jednostavno analizirati u Three.js editoru.



Slika 15 – Prikaz Three.js online editora

3.2 Transformiranje objekata

Prije nego što animiramo objekte, moramo znati kako transformirati objekte. Postoje 4 svojstva za transformiranje objekata:

- **position**
- **scale**
- **rotation**
- **quaternion**

3.2.1 Pomicanje objekata

Objekte možemo pomicati po 3 koordinate: x, y i z. Brojčana udaljenost od 1 može u Three.jsu biti bilo što. Npr. može biti: 1 kilometar, 1 metar, 1 centimetar i sl.
Pozicije objekata u kodu možemo pisati bilo gdje, no ne možemo pisati nakon `rendera` scene, drugim riječima, ništa se neće promijeniti ako mi uslikamo sliku (`render`) te zatim promijenimo poziciju. Umjesto da za pozicije pišemo npr. **mesh.position.x**, **mesh.position.y** itd. možemo upotrijebiti **mesh.position.set()** i kao parametre unijeti redom x, y i z.
Pozicioniranje objekata u prostoru nije lako, kako bismo si olakšali posao možemo koristiti klasu unutar Three.jsa koja se zove `Axes Helper`. Ovim pomagalom možemo prikazati pomoćne crte u boji (crvena, zelena, plava) za svaku os. Crvena boja predstavlja pozitivnu x os, zelena predstavlja pozitivnu y os i plava predstavlja pozitivnu z os.

3.2.2 Skaliranje objekata

Skaliranje objekata jako je jednostavno i ima 3 svojstva: x, y i z. Početna zadana vrijednost svake osi je 1. Za lakše postavljanje vrijednosti skaliranja objekata možemo koristiti **mesh.scale.set()**.

3.2.3 Rotacija objekata

Za razliku od pomicanja i skaliranja objekata, rotacija objekata je malo teža. Rotaciju objekata možemo izvoditi pomoću `rotation` ili pomoću `quaternion`. Ako trebamo zarotirati neki objekt za pola, možemo to učiniti koristeći matematičku PI vrijednost s **Math.PI**.

Moramo biti oprezni pri rotaciji objekata po osi zato što možemo ujedno zarotirati i drugu os, zadana rotacija ide redom: x, y, z. Ne poštujući ovaj redoslijed može se desiti da nam osi više 'ne rade'. Ovakav problem zove se `gimbal lock`. Ovaj problem može se riješiti koristeći **mesh.rotation.reorder('YXZ'),** stavljajući ovu liniju koda na sam početak prije mijenjanja rotacija. Ovo je razlog zašto većina 3D softvera koriste **Quaternion**.

3.2.4 Quaternion

`Quaternion` isto predstavlja rotaciju, no više u matematičkom smislu. `Quaternion` se ažurira nakon što promijenimo `rotation`.

3.2.5 Look at

Jedna od zanimljivih značajki Three.jsa je metoda **lookAt()** kojom se može postići gledanje u određeni objekt. **lookAt()** metodu jednostavno je koristiti.

3.2.6 Scene graph

Ponekad nam se može desiti da imamo više objekata unutar jednog velikog objekta te da želimo promijeniti veličinu velikog objekta, recimo, smanjiti ga. Možemo to učiniti, međutim ostali objekti će ostati iste veličine. Ovaj problem možemo riješiti tako da stavimo objekte unutar jedne grupe i pomoću `position`, `rotation` (ili `quaternion`) i `scale` promijenimo vrijednosti objektima. Za to će nam biti potrebna klasa `Group`.

3.3 Animacije

Animacije u JavaScriptu rade na način: pomakni objekt > uslikaj fotografiju > pomakni objekt > uslikaj fotografiju. Pri animaciji dolazi do razmjene više slika koje se najčešće prikazuju u brzini od 60 sličica po sekundi. Većina monitora radi na 60 sličica u sekundi. Animacije moraju izgledati isto bez obzira na brzinu broja sličica u sekundi.

Za napraviti animaciju moramo ažurirati objekte i napraviti `render` na svakome `frameu`. To ćemo učiniti u funkciji i pozvati funkciju sa: **window.requestAnimationFrame()**
Razlog pozivanja funkcije **requestAnimationFrame()** je pozivanje funkcije osigurane na sljedećem `frameu`. Pozivati ćemo istu funkciju na svakom `frameu`.

Na žalost, što je veći `framerate`, to je brža rotacija, pa se dešava da oni koji imaju monitore više od 60 Hz, objekti im se npr. vrte brže. No ovaj problem se može riješiti na više načina Jedan od načina je saznati proteklo vrijeme od početka animacije uspoređujući trenutno vrijeme s prijašnjim vremenom. Tako ćemo dobiti objekte koji se npr. vrte istom brzinom bez obzira na `framerate`.

3.3.1 Biblioteka za animaciju

Ako želimo imati veću kontrolu, stvoriti vremenske crte, itd. možemo koristiti biblioteku poput **GSAP**. Ovu biblioteku možemo u svoj projekt dodati putem terminala s: **npm install –save gsap@3.5.1**

3.4 Kamere

U Three.js biblioteci postoji više kamera poput:

- **ArrayCamera**
- **Camera**
- **CubeCamera**
- **OrthographicCamera**
- **PerspectiveCamera**
- **StereoCamera**

- `Camera` - apstraktna klasa, ne bismo ju trebali koristiti izravno.

- `ArrayCamera` - prikazuje scenu s više kamera na određenim područjima `rendera`, ovakvo nešto se može upotrijebiti u tkz. `splitscreen` igricama gdje je zaslon podijeljen npr. na dva dijela.

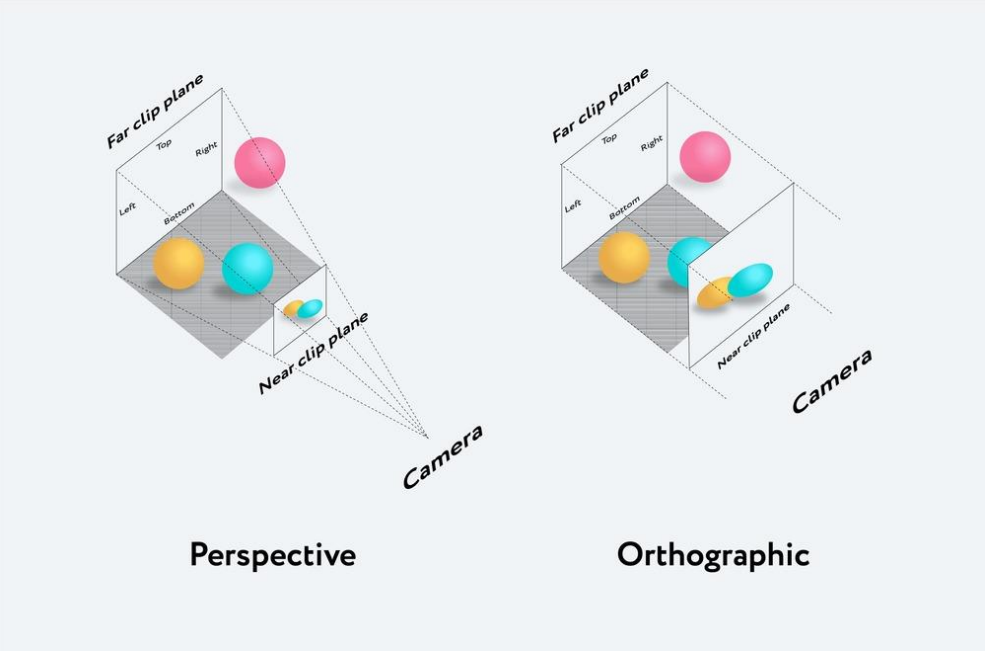
- `StereoCamera` - prikazuje scenu kroz dvije kamere koje oponašaju oči kako bi se stvorio paralaksni efekt, najčešće se koristi za VR naočale, crvene i plave naočale i sl.

Dakle jedno oko gleda u jednu kameru a drugo oko u drugu, tako se postiže dojam dubine.

- `CubeCamera` – ova kamera napravi 6 `rendera`, svaki okrenut u drugom smjeru. Ovakva se kamera najviše koristi za `environment mape`, refleksije i za `shadow mape`.

- `OrthographicCamera` – ova kamera će prikazati scenu bez perspektive, drugim riječima, ako je objekt jako udaljen od kamere imati će istu veličinu kao i kad je blizu kamere. Parametri ove kamere su: `left`, `right`, `top`, `bottom`, `near`, `far` koji označavaju koliko daleko kamera može gledati u svakome smjeru.

- `PerspectiveCamera` – ova kamera će prikazati scenu s perspektivom. Prvi parametar ove kamere je `field of view` koji označava kut vida, drugi parametar je `aspect ratio` koji označava širinu `rendera` podijeljenu s visinom `rendera`. Zadnja dva parametra su: `near` i `far` koja označavaju koliko blizu i koliko daleko kamera može vidjeti. Svaki objekt bliži od `near` ili dalji od `far` neće se prikazati.
- Ne preporuča se korištenje ekstremnih vrijednosti poput: 0.0001 i 9999999 zato što ćemo dobiti grešku koja se zove `z-fighting`.



Slika 16 – Vizualni prikaz razlike između Perspective i Orthographic kamere

3.5 Kontrole

Imamo više različitih kontrola:

- **DeviceOrientationControls**
- **DragControls**
- **FirstPersonControls**
- **FlyControls**
- **OrbitControls**
- **PointerLockControls**
- **TrackballControls**
- **TransformControls**

- `DeviceOrientationControls` - automatski će dohvatiti orijentaciju uređaja (ako to uređaj dopušta), operacijski sustav i preglednik te rotirati kameru u skladu s tim. Koristi se najčešće kod pametnih telefona za VR.

- `DragControls` - nema nikakve veze s kamerom. Ovom kontrolom možemo pomicati objekte.

- `FirstPersonControls` - jako sličan `FlyControlsu` ali s fiksnom gornjom osi.

- `FlyControls` - omogućuje pomicanje kamere nalik vožnji svemirskog broda. Može se rotirati oko sve 3 osi, ići naprijed i nazad.

- `OrbitControls` - često se koristi za okretanje kamere oko 3D objekata, možemo zumirati i odzumirati te desnim klikom miša pomicati kameru u stranu.

- `PointerLockControls` - koristi se kada se želi imati kontrolu pomicanja kamere sa pokazivačem miša bez da je pokazivač miša vidljiv. `PointerLockControls` koristi `pointer lock JavaScript API`.

- `TrackballControls` - poput `OrbitControlsa`, no bez okomitog ograničenja kuta.

- `TransformControls` - nema nikakve veze s kamerom. Njime možemo izraditi editor s vidljivim osima nalik onome u softveru `AutoCad`.

3.6 Fizika

Možemo napraviti svoju fiziku s matematikom i rješenjima poput `Raycastera`, no ako želimo realističnu fiziku s napetošću, trenjem, odskakivanjem, ograničenjima, zaokretima itd., bolje je koristiti biblioteku. Većinom se sve može napraviti u Three.js `svijetu` no za fiziku nam treba još jedan `svijet` kojega ne vidimo. Implementacija fizike nije teška, no organizacija koda je.

3.6.1 Biblioteke za fiziku

Postoji puno biblioteka koje omogućuju fiziku u Three.jsu. Moramo izabrati između korištenja 3D biblioteke ili 2D biblioteke.

Najpopularnije 3D biblioteke za fiziku su:

- **Ammo.js**
- **Cannon.js**
- **Oimo.js**

Najpopularnije 2D biblioteke za fiziku su:

- **Matter.js**
- **P2.js**
- **Planck.js**
- **Box2D.js**

Postoji i mogućnost kombiniranja Three.jsa s bibliotekom za fiziku poput `Physijs`. Najčešće korištena biblioteka je Ammo.js, međutim Cannon.js biblioteku lakše je implementirati i razumjeti.
Komponenta našeg računala koji omogućuje fiziku je procesor. Trenutno, sve se radi od strane jedne dretve na našem procesoru pa se dretva može brzo opteretiti.
Rješenje je koristiti tkz. `workers`. `Workers` omogućuju postavljanje dijela koda u drugu dretvu kako bi rasteretili 'teret'. Na ovaj se način mogu poboljšati performanse. Uporabom fizike, iskustvo se drastično može poboljšati, no nije lagano i proces implementacije je dug.

3.7 Puni zaslon i promjena veličine zaslona

3.7.1 Puni zaslon

Ako npr. radimo web videoigru poželjno je da korisnik može ući u tkz. `fullscreen mod`. To se može jednostavno postići, uz pomoć dvoklika mišem:

```
window.addEventListener('dblclick', () =>
{
      if(!document.fullscreenElement)
      {
      canvas.requestFullscreen()
      }
      else
      {
      document.exitFullscreen()
      }
})
```

Moramo imati na umu da ovaj kod možda neće raditi na web pregledniku poput Safaria, no srećom se taj problem može riješiti.

Rezoluciju `canvasa` možemo postaviti koju god želimo, no najčešće se u praksi koristi `canvas` preko cijelog zaslona. `Canvas` preko cijelog zaslona možemo postići uz pomoć: **window.innerWidth** i **window.innerHeight**, no najvjerojatnije ćemo primijetiti da su se naokolo pojavile margine. Ovaj slučaj se dešava zbog zadanih stilova preglednika.

Možemo ovo riješiti na više načina. Jedan od njih je da unesemo dolje prikazani kod u CSS datoteku:

```
*
{
      margin: 0;
      padding: 0;
}
```

Drugi način na koji se to može riješiti (ako nam je klasa `canvasa webgl`):

```
.webgl
{
      position: fixed;
      top: 0;
      left: 0;
      outline: none;
}
```

Ako korisnik `scrolla` moguće je da će uspjeti vidjeti nešto što ne bi trebao ispod stranice. Kako bismo to riješili i bili sigurni da korisnik ima onemogućen `scroll`, dodati ćemo `overflow`**:** `hidden` na `html` i `body` u CSSu.
No ako pokušamo povećati i smanjiti prozor web preglednika primijetiti ćemo da `canvas` ostaje na istome mjestu. Ovo možemo riješiti tako što ćemo 'slušati' kada će prozor biti smanjen ili povećan:

```
const sizes =
{
      width: window.innerWidth,
      height: window.innerHeight
}

window.addEventListener('resize', () =>
{
      sizes.width = window.innerWidth
      sizes.height = window.innerHeight

      camera.aspect = sizes.width / sizes.height
      camera.updateProjectionMatrix()

      renderer.setSize(sizes.width, sizes.height)
})
```

3.7.2 Pixel ratio

Ponekada možemo vidjeti nejasni obris i efekt stepenica na rubovima 3D objekata. Ovo se dešava zbog toga što testiramo na zaslonu s omjerom piksela većim od 1. Omjer piksela odgovara tome koliko fizičkih piksela imamo na zaslonu za jednu jedinicu piksela na softverskom dijelu. Prije nekoliko godina, svi zasloni imali su `pixel ratio` od 1 te ako bismo pogledali malo bliže u zaslon, vidjeli bismo piksele. Danas proizvođači zaslona rade `pixel ratio` od 3 ili više.

Kako bismo dobili trenutni `pixel ratio`, možemo koristiti **window.devicePixelRatio**. U praksi se najčešće postavlja `pixel ratio` u vrijednosti od 2. Vrijednost možemo postaviti unutar `resize event listenera`.

3.8 Debug UI

Moramo moći prilagoditi i ispraviti pogreške. Recimo da radimo scenu u kojoj imamo objekte na kojima želimo isprobavati boje, napisali bismo kod, provjerili rezultat, promijenili bismo kod, opet provjerili rezultat itd. Ovakav način rada uzima jako puno vremena ali u tome nam može pomoći `Debug UI`. Možemo napraviti vlastiti `Debug UI` ili koristiti jednu od biblioteka poput: `datGUI`, `control panel`, `ControlKit`, `Guify`, `Oui`.

Od svih ovih biblioteka, najpopularniji je `datGUI`.

Slika 17 – Primjer izgleda datGUIa

3.8.1 Implementacija datGUIa

- Za instalaciju možemo u terminalu upisati: **npm install --save dat.gui**
- Ovu biblioteku možemo uvesti s: **import * as dat from `dat.gui`**
- Instancirati ju možemo sa: **const gui = new dat.GUI()**

Nakon ovoga, u gornjem desnom kutu ekrana pojaviti će nam se tkz. `panel`.
`Panel` je u početku prazan, u njega možemo dodati elemente poput: `range`, `color`, `text`, `checkbox`, `select`, `button`, `folder` i sl.

`Range` - za brojeve s minimalnom i maksimalnom vrijednošću
`Color` - za boje s raznim formatima
`Text` – za jednostavan tekst
`Checkbox` - za logičke vrijednosti (`true` ili `false`)
`Select` - iz izbora popisa vrijednosti
`Button` - za pokretanje funkcija
`Folder` - za organiziranje ploče kada imamo više elemenata

`Panel` možemo sakriti pritiskom na tipku '**H**'. Ako želimo da je sakriven već u početku možemo to učiniti s: **gui.hide()**. Nakon ove funkcije, ako želimo prikazati `panel` to možemo učiniti dvoklikom tipke '**H**'.

3.9 Savjeti za bolje performanse

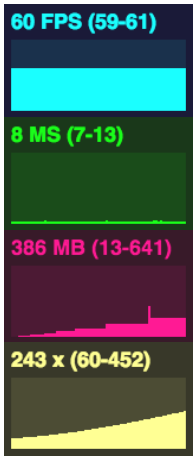
Trebali bismo ciljati na minimalno 60 `fpsa` tj. sličica u sekundi. Mogu postojati dva glavna ograničenja:

- Procesor
- Grafička kartica

Testirati izrađenu web stranicu putem više uređaja i web preglednika dobra je praksa. Isto tako treba pripaziti i na `težinu` web stranice.

3.9.1 Nadgledanje

Pošto trebamo mjeriti performanse potrebno ih je nadgledati. Možemo koristiti `JavaScript FPS meter` poput **`stats.jsa`**.



Slika 18 – Izgled `stats.jsa` u primjeni

Smanjivanjem ili povećavanjem veličine web preglednika moguće je primijetiti promjenu `frameratea`.

Jedan od dobrih Chrome dodataka je **Spector.js** koji nam omogućuje uvid u što Three.js crta. `Renderer informations` kojeg možemo pozvati jednostavnim `console.logom` je još jedan od načina kako dobiti dodatne informacije.

On nam govori što je sve spremljeno u memoriju poput: geometrija, tekstura, programa, poziva, broj sličica u sekundi, linija, točaka, broj trokuta itd.

3.9.2 Korisni savjeti

- Izbjegavati stvari koje su teške poput npr. dugačkih petlji, puno objekata, velikih polja i sl.

- Izbjegavati svjetla u Three.jsu, umjesto njih bilo bi dobro koristiti `baked lights` ili `cheap lights` poput: `AmbientLight`, `DirectionalLight` i `HemisphereLight`.

- Izbjegavati sjene u Three.jsu, preporuča se korištenje `baked shadowsa`.

- Kako bi pristajale perfektno sa scenom, `ShadowMape` je bolje što više suziti.

- Promijeniti veličinu teksturama dobro je zato što teksture zauzimaju jako puno prostora u memoriji grafičke kartice.

- Smanjivanje rezoluciju teksture na minimum, zadržavajući pristojan izgled, dobra je praksa.

- Korištenjem pravog formata može se znatno smanjiti vrijeme učitavanja. Preporuča se korištenje formata poput: **.jpg** ili **.png**. Možemo koristiti i online alate poput **`TinyPNG`** kako bismo još više smanjili težinu slike.
  Postoji i tkz. `basis` format koji je kao i .jpg i .png, no kompresija je puno jača i format može biti lakše očitan od strane grafičke kartice. Na žalost, teže ga je generirati i dosta se izgubi na kvaliteti pri kompresiji slike.

- Umjesto da koristimo obične geometrije možemo koristiti tkz. `buffer` geometrije koje su bolje što se tiče performansi. Ažuriranje vrhova geometrije nije dobro za performanse pogotovo ako imamo puno geometrija.

- Dobra je preporuka koristiti tkz. `cheap` materijale. Neki materijali poput `MeshStandardMaterial` ili `MeshPhysicalMaterial` trebaju više resursa nego materijali poput `MeshBasicMaterial`, `MeshLambertMaterial`, `MeshPhongMaterial`.

- Dobra je preporuka koristiti tkz. `low poly` modele, što manje poligona to bolje.

- Ako model ima jako puno detalja s jako kompleksnim geometrijama, dobra je preporuka koristiti `Draco` kompresiju. GZIP je kompresija koja se događa na strani servera. Većina servera ne `gzipira` datoteke poput: .glb, .gltf, .obj.

- Vidno polje kamere dobro je smanjiti jer se tako `rednerira` manje objekata. Isto tako dobro je smanjiti `near` i `far` svojstva kamere.

- Dobro je limitirati `pixel ratio` na vrijednost od 2.

- Neki od uređaja imaju mogućnost prebacivanja između različitih upotreba grafičke kartice ili procesora. Pri instanciranju `WebGLRenderera` pomoću specificiranja `powerPreference` svojstva, možemo Three.jsu 'reći' koja snaga je potrebna.

- `Antialis` je dobro postaviti samo u slučaju ako se objekt ne vidi dobro, inače ga ne treba postavljati.

- `IF` uvjete dobro je izbjegavati.

Primjenjujući ove savjete ne mora uvijek biti dobra ideja, svaki projekt imati će svoje zahtjeve.

4. Izrada Three.js projekta

Web Stranica: https://jastuciu3d.000webhostapp.com/
GitHub: https://github.com/sandiivanusec/Jastuci-u-3D

4.1 Uvod u projekt

Ideja za ovaj projekt proizašla je iz problema s kojim se često susreću `YouTuberi`. Naime kako bi `YouTuberi` mogli zarađivati od izrade svojih videozapisa nudi im se više mogućnosti poput:

- Reklama
- Postavljanja reklamnih `URLova`
- Pregledavanja proizvoda ili softvera
- Sponzora
- Prodajom vlastitih proizvoda
- Donacija

Česti izvor zarade imaju od prodaje vlastitih proizvoda na kojem se nalazi najčešće logotip samog `YouTubera` koji ga predstavlja. Njihovi proizvodi mogu biti npr.:

- Kape
- Majice
- Jakne
- Jastuci

Zadnja i najmanja prodavana stvar na ovom popisu su jastuci. `YouTuberi` nemaju preveliku prodaju od jastuka pa većina i odustaje od prodaje takvog proizvoda.
Ideja ovog projekta bila je napraviti web stranicu koja će kupcu omogućiti pregled jastuka u 3Du te ga na neki način 'privući' da se zainteresira za kupnju jednog od jastuka. Ideja je bila to napraviti na drugačiji način nego što to inače web stranice nude, na minimalističan i zabavan način kako se korisnik ne bi 'izgubio' na web stranici te kako bi se 'zabavio' okrećući jastuke.

4.2 Struktura projekta

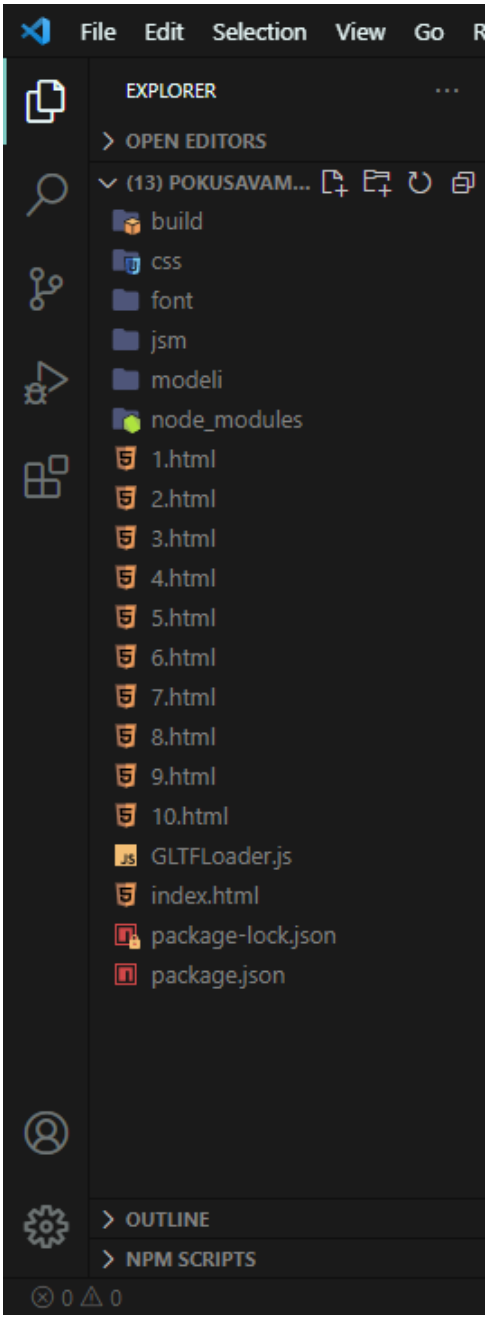
Nakon instalacije Three.js biblioteke imali smo mogućnost odabira 3 vrste Three.jsa u `build` mapi. Odabrali smo **three.module.js** kako bismo lakše mogli uvoziti dijelove Three.jsa koji su nam bili potrebni.

Jedini font koji smo koristili je 'Iceland' font u .json formatu, kojeg smo radi bolje preglednosti smjestili u mapu nazvanu `font`.

Mape poput `jsm`, `node_modules` i `GLTFLoader.js` datoteke koje dolaze uz instalaciju Three.js biblioteke 'izvukli' smo iz mapa u tkz. `root` kako bismo lakše navigirali kada trebamo uvesti neki od posebnih dijelova Three.jsa.

U mapu `modeli` smjestili smo svih 10 različitih modela jastuka koje smo izradili pomoću softvera Blender.

U `root` smo postavili različite veličine `favicona` za različite uređaje na kojima će se web stranica izvoditi.



Slika 19 – Struktura Three.js projekta

4.2.1 CSS

U projektu smo koristili CSS kako bismo ljepše oblikovali web stranicu. CSS smo koristili za tri stvari: za `buttone`, za sam prozor web stranice i za cijenu. Postoje 2 vrste `buttona` koje smo koristili a to su: `button` 'POGLEDAJ' za početak ulaska u pregled jastuka i `buttoni` za navigaciju između različitih jastuka. Prelaskom miša nad svakim `buttonom`, pokazivač miša se pretvara iz uobičajnog u oblik ruke s kažiprstom. Obje vrste `buttona` stavili smo u posebne klase kako bismo ih kasnije mogli zajedno i lakše pozicionirati. Pojavio se problem bijelih crta oko samog prozora tj. scene, no to smo uspjeli riješiti pomoću svojstava za `body` i `.webgl`. Iznad svakog jastuka pozicionirana je cijena kako bi uvijek bila vidljiva bez obzira na veličinu zaslona pametnih uređaja.

4.2.2 Početna stranica

Glavna datoteka od koje sve počinje u ovome projektu i u kojem je pisana velika većina JavaScript koda za ovaj projekt je **index.html**. Naravno, kod se mogao pisati i u zasebnoj datoteci **index.js**. Kao naslov ovoga projekta a i same HTML stranice odabran je jednostavan naziv **'Jastuci u 3D'**. Kako bi se iskoristili razni stilovi u HTML datotekama stvorena je **style.css** datoteka koja direktno komunicira sa **index.html** datotekom. Naslov i povezivost htmla i cssa zapisani su u `head` dijelu HTML datoteke.

Velika većina JavaScript koda pisana je unutar HTMLa. Na početnoj web stranici nalazi se HTML `button` naziva 'POGLEDAJ'. Klikom na ovaj `button` prelazi se na prvu stranicu s prvim jastukom nazvanom **1.html**.

Kako bi se moglo započeti pisanjem Three.js tj. JavaScript koda bilo je potrebno uvesti Three.js biblioteku iz mapa koje se dobiju prethodnom instalacijom Three.jsa. Osim same Three.js biblioteke uveli smo i `GLTF Loader` koji nam omogućuje uvoz vanjskih 3D modela tj. modela koji nisu izrađeni pomoću Three.jsa. Vanjski 3D modeli za ovaj projekt izrađeni su u 3D softveru Blender i izvezeni u .gltf formatu, specijaliziranom formatu za učinkovitu isporuku i učitavanje 3D sadržaja. Uveli smo i tkz. `Orbit Controls` koji nam služi za slobodno okretanje kamere pomoću računalnog miša. Preostale dvije stvari koje smo uvezli odnose se na tkz. `RectAreaLight` svjetla a to su: `RectAreaLightHelper` i `RectAreaLightUniformsLib`. Potrebne su obje stvari kako bi ovakvo svjetlo radilo.

Tri stvari bez kojih ne možemo u Three.jsu su `renderer`**, kamera** i **scena**. Kao `renderer` koristili smo `WebGLRenderer`. Prilikom vrtnje objekata pojavljivale su se neravne crte na rubovima objekata. Kako bismo riješili problem i prikazali rubove objekata pri vrtnji prirodnijima, postavili smo `antialias` na `true`. Kao kameru koristili smo `PerspectiveCamera`. Kao scenu stvorili smo najjednostavniju scenu. U našem kodu postoji funkcija **init()** kojom smo postavili pozadinu scene u crnu boju, namjestili kameru na točno određenu poziciju i postavili veličinu `renderera`. Sve ovo postavlja se nakon svakog osvježavanja stranice. Funkcijom **setLight()** stvorili smo `AmbientLight`, nježno svjetlo bijele boje za našu scenu.

U početnoj sceni koristili smo jednostavnu podlogu bijele boje s malom prozirnošću kako bismo kasnije mogli postići željeni efekt odbijanja svjetla o tu istu podlogu. U scenu smo dodali dva `RectAreaLight` identična svjetla ljubičaste boje: jedno s gornje lijeve strane i jedno s donje desne strane. Oba svjetla su jačine vrijednosti 10. Kako bi ih postavili na željeno mjesto, svjetla je bilo potrebno u početku rotirati i pozicionirati.

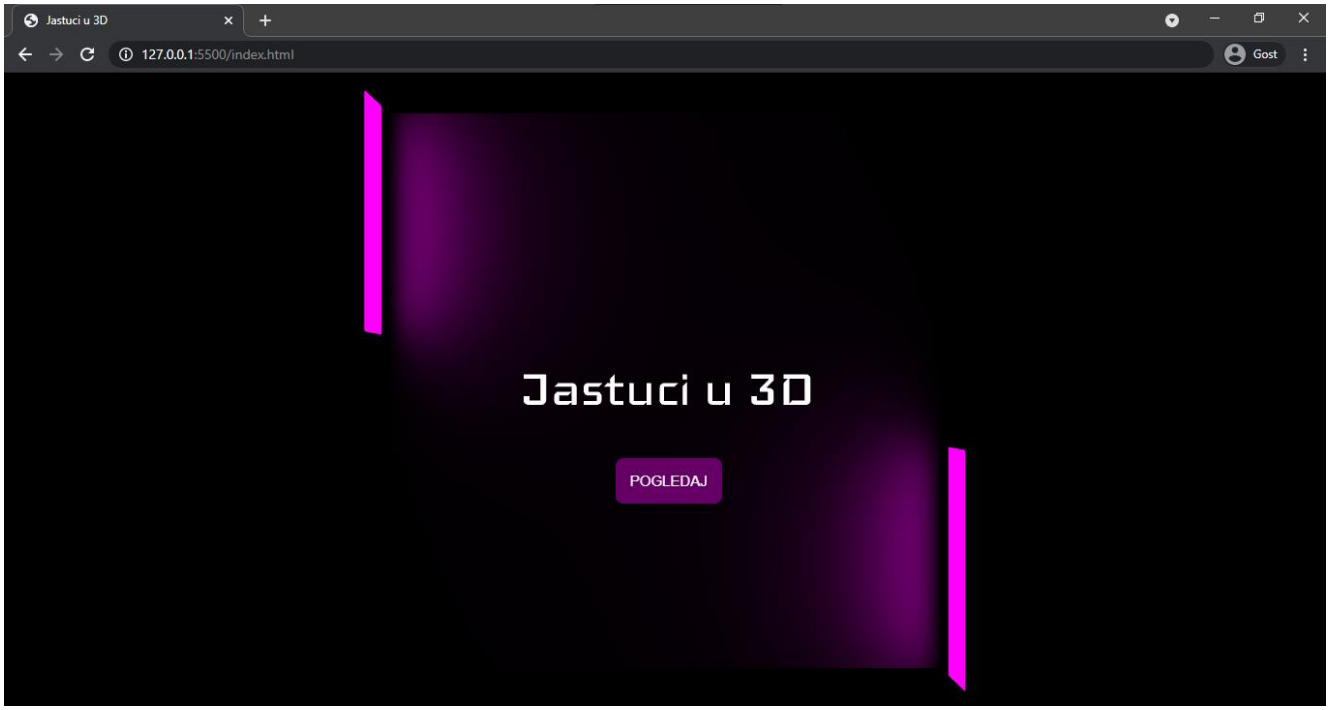
Za tekst smo preuzeli font **Iceland** s `Google Fonts` web stranice te ga konvertirali u .json format pomoću web aplikacije `Facetype.js`. Kako bi Three.js mogao koristiti bilo koji font koji nije vezan uz samu biblioteku, bila je potrebna konvertizacija fonta. Učitali smo font pomoću tkz. `Font Loadera`. Tekst koji je ovim fontom napisan u prvoj sceni je 'Jastuci u 3D'. Tekstu smo postavili veličinu i visinu. Kako za tekst u ovome slučaju nismo htjeli da reagira na ljubičasta svjetla, postavili smo materijal `MeshBasicMaterial`. Ako bismo postavili npr. `MeshStandardMaterial,` tekst bi bio osvjetljen ljubičastom bojom te bi bio nepregledan. Pozicionirali smo tekst na samu sredinu podloge.

Kako prva scena ne bi bila statička, omogućili smo okretanje kamere pomoću `Orbit Controls`. Pošto se za početnu scenu nemamo potrebu rotirati punim krugom oko podloge, teksta i svijetla, limitirali smo koliko korisnik može okretati kameru. Limitaciju smo postavili za minimalno/maksimalno horizontalno okretanje, minimalno/maksimalno vertikalno okretanje i minimalno/maksimalno zumiranje.
Ovim limitacijama onemogućili smo korisniku da vidi stvari koje ne bi trebao vidjeti.
Okretanje kamere nije bilo 'glatko', kako bismo postigli 'glatkoću klizanja' kamere omogućili smo tkz. `Damping`.

Korisnik naše web stranice može koristiti web stranicu na različitim pametnim uređajima što znači da veličina stranice mora biti prilagodljiva. Kako bismo riješili ovaj problem, postavili smo konstantne veličine za visinu i širinu naše scene koje poprimaju visinu i širinu unutarnjeg prozora u kojem se web stranica nalazi. Bilo je potrebno postaviti tkz. `Event Listener`, 'slušača' koji se pokreće kada se dogodi promjena veličine prozora. Kada korisnik počne mijenjati veličinu prozora ili se veličina prozora sama promjeni zbog određene veličine ekrana različitih pametnih uređaja, tada se ažuriraju vrijednosti visine i širine unutrašnjeg prozora, vidno polje kamere i `renderer`.

Kako bismo onemogućili automatsku rotaciju kamere koju koristimo na ostalim stranicama te ažurirali kontrole kamere i sam `renderer`, bila nam je potrebna funkcija **animate()**.
Na ovoj stranici omogućen je `fullscreen` način rada.

Ovu prvu scenu zamislili smo u mraku sa malom jačinom dvoje svjetla koja će 'pokazati put' gdje novi korisnik treba gledati.



Slika 20 – Prikaz početne stranice projekta

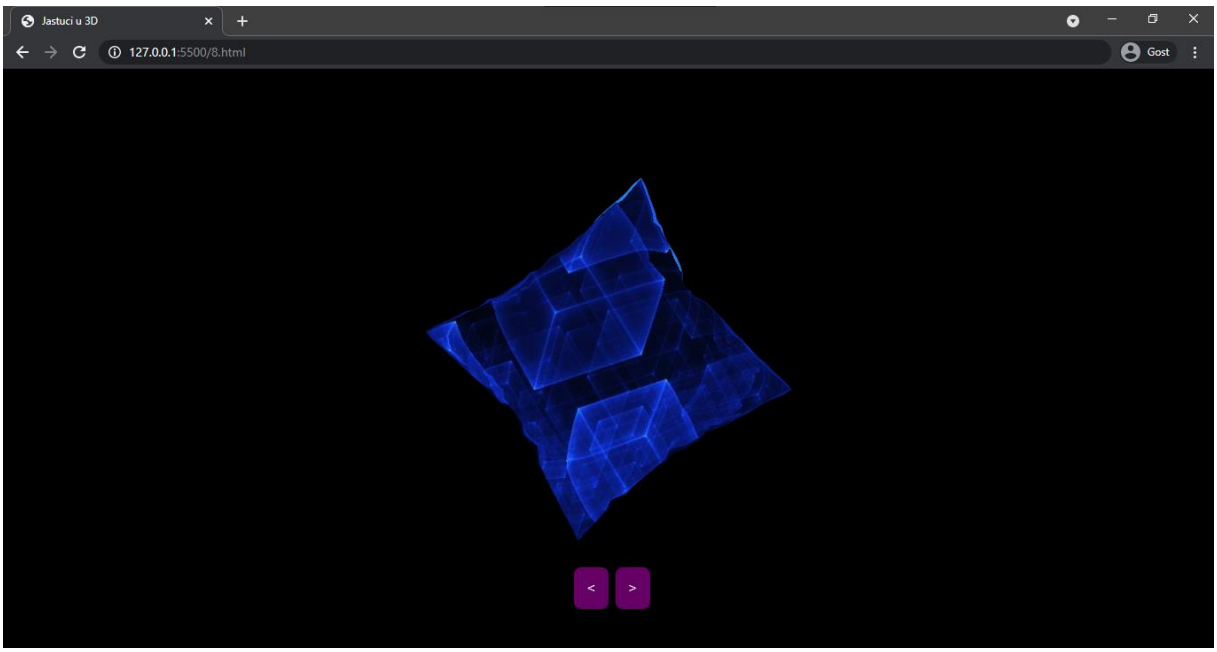
Kako se ne bi korisniku omela pažnja, te kako ne bi prešao na neku od drugih konkurentnih stranica, početna je stranica namjerno napravljena na minimalističan način. Drugim riječima, ova scena pokušava korisnika privući da klikne na gumb 'POGLEDAJ' kako bi razmotrio jastuke. Kako bi korisnik odmah uočio gdje treba kliknuti mišem, u sceni je 'namjerno pušten' malo veći razmak između svjetla i teksta. Isto tako, oko podloge ne nalazi se ništa, kako bi fokus korisnika bio još jači.

4.2.3 Preostale stranice

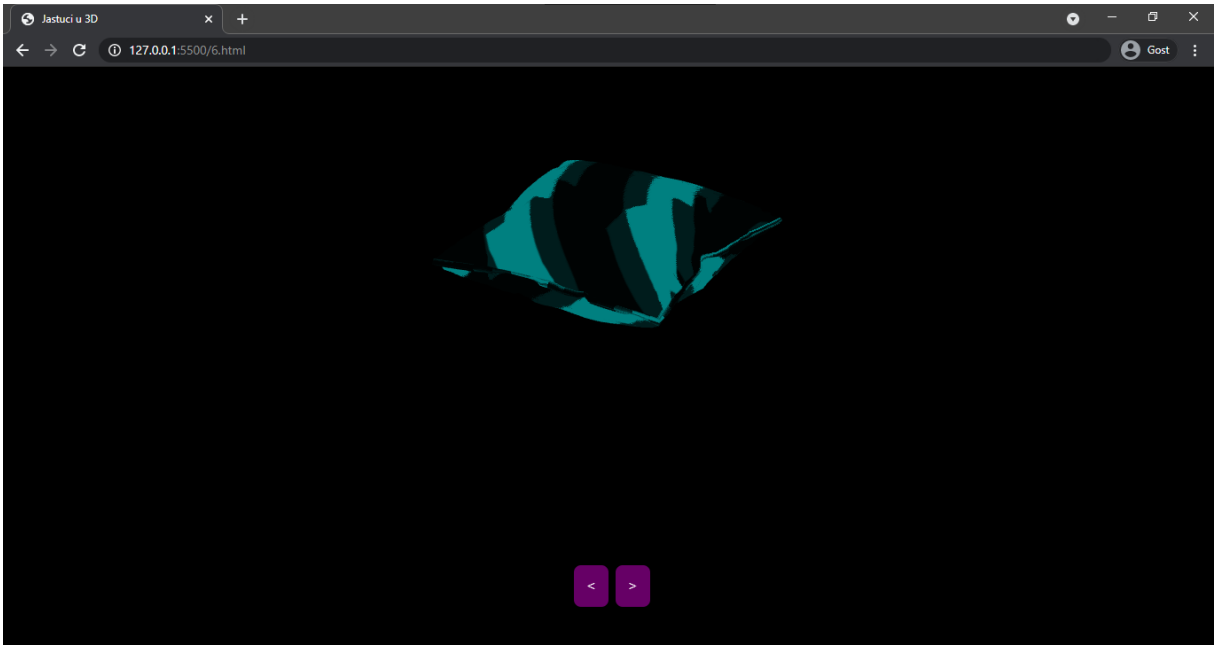
Na svakoj se od ovih stranica nalazi po 2 gumba za prelazak na prethodnu i sljedeću stranicu. Kako nebismo previše opteretili grafičku karticu uređaja na kojem se web stranica gleda, u svakoj od stranica uvezen je samo po jedan jastuk. Zato što se neki jastuci ne vide kako treba zbog njihove tamne boje, uveli smo svjetlo `DirectionalLight`. Za razliku od početne stranice ovdje smo limitirali micanje kamere samo što se tiče zumiranja, dakle korisnik može do određene razine zumirati i odzumirati. Omogućili smo korisnicima potpunu kontrolu nad okretanjem jastuka, kako bi ih korisnici mogli razgledati sa svih strana. U funkciji **animate()** omogućili smo automatsku rotaciju jastuka.

Svaki jastuk 'namjerno' je postavljen u ležećem položaju kako korisniku ne bi bio vidljiv u potpunosti, smatramo da će korisnik instiktivno shvatiti da jastuk može povući i okretati, bez da smo mu to rekli. Na sličan smo način instiktivno objasnili korisniku zumiranje i odzumiranje u smjeru jastuka tako da smo jastuk malo udaljili. U scenama jastuka nema ničeg drugog osim jastuka i gumbića za prelazak na prethodni ili sljedeći jastuk, na taj se način korisnik u potpunosti može fokusirati na promatranje jastuka. Korisnik ima mogućnost zumiranja i odzumiranja (pomoću kotačića na mišu), rotiranja oko jastuka (pomoću lijeve tipke miša) te pomicanja u prostoru (pomoću desne tipke miša). Kao i na početnoj stranici, tako je i na ovima omogućen `fullscreen` način rada. Na svakoj od preostalih stranica prikazana je cijena za svaki pojedini jastuk.

Slika 21 – Prikaz jastuka crno bijele boje



Slika 22 – Prikaz jastuka plave boje



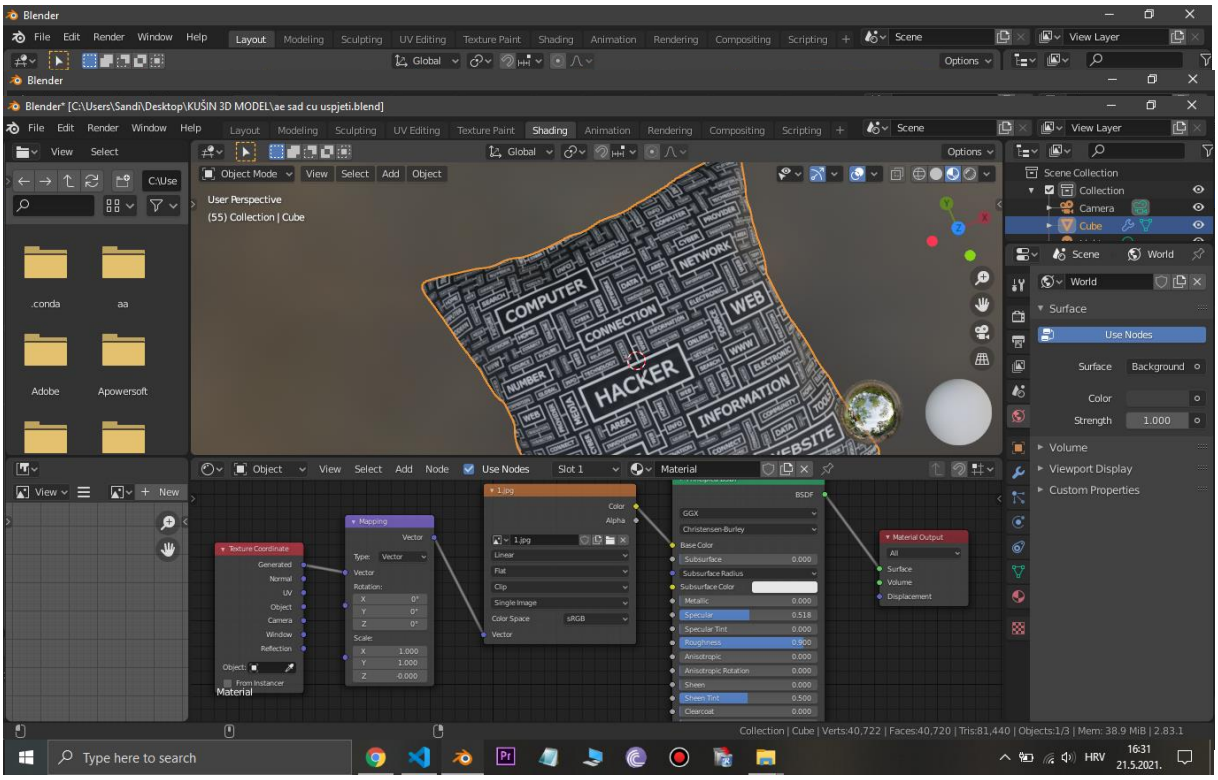
Slika 23 – Prikaz jastuka zeleno crne boje

Slika 24 – Prikaz jastuka bijelo crne boje

4.3 Izrada, tekstura i izvoz 3D modela

4.3.1 Izrada modela

- Za izradu 3D modela jastuka koristili smo besplatni 3D softver za modeliranje i animaciju Blender.
- Modeliranje smo započeli jednostavnom kockom kojoj smo promijenili oblik u plohu. Plohu smo podijelili na 30 identičnih dijelova horizontalno i vertikalno.
- U odjeljku `Physics` odabrali smo vrstu tkanine svilu. Gravitaciju smo postavili na vrijednost 0. Oko jastuka stvorili smo tkz. polje sile kojem smo postavili jačinu na 150.
- Pokrenuli smo simulaciju tipkom `Play` te čekali da sila uzdigne plohu do poprimanja oblika sličnome jastuku. Kada smo bili zadovoljni veličinom, prekinuli smo simulaciju.
- Kako bismo dobili više realističniji oblik jastuka koristili smo funkciju `Shade Smooth` kojom smo izgladili oblik jastuka.
- Kako bi jastuk izgledao još realističnije, proširili smo rubove jastuka te ih još malo izvukli.



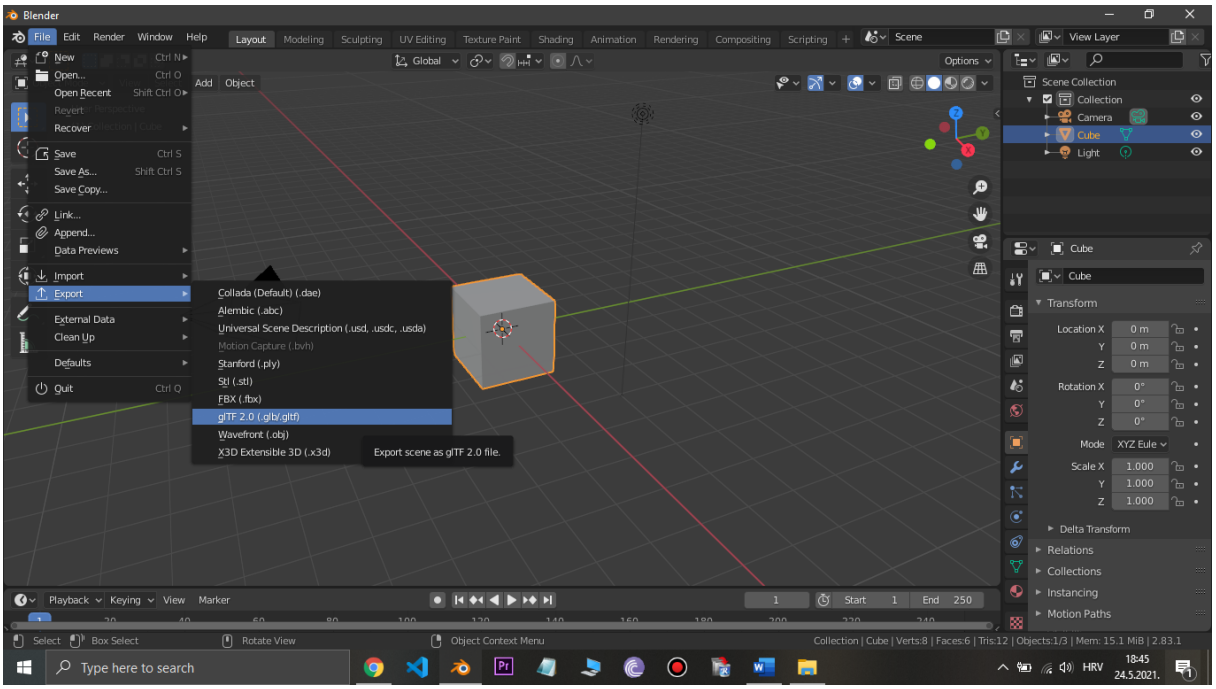
Slika 28 – Proces postavljanja teksture na jastuk

4.3.2 Tekstura modela

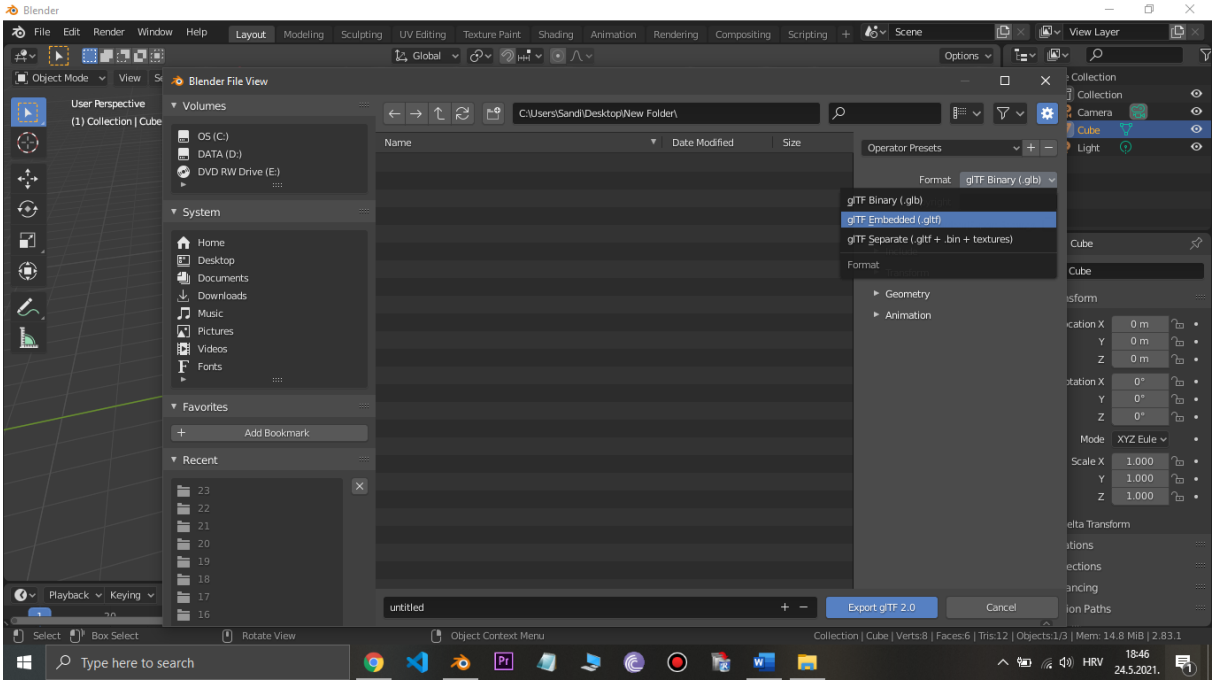
- Za teksturu jastuka morali smo se prebaciti u `Shading` dio Blendera.
- Iz izbornika u donjem dijelu dodali smo `Texture Coordinate`, `Mapping` i `Image Texture`.
- U `Image Texture` odabrali smo različite slike, za svaki jastuk posebno.
- U `Mapping` dijelu smo mijenjali lokaciju, rotaciju i skalu slike.

4.3.3 Izvoz 3D modela

- U Blenderu smo s padajućeg izbornika izabrali sljedeće: **File -> Export -> glTF 2.0**.
- Zatim smo kao format izabrali `glTF Embedded`.
- Klikom na `Export glTF 2.0` izvezli smo 3D model jastuka na mjesto našeg računala koje smo prethodno definirali.



Slika 29 – Prikaz koraka izvoza 3D modela



Slika 30 – Prikaz koraka odabira formata za izvoz 3D modela

5. Zaključak

Three.js je odlična biblioteka za sve one koji tek započinju, kao i za one koji su iskusni u izradi 3D web stranica. Biblioteka znatno olakšava programiranje u usporedbi s WebGLom. Three.jsom može se gotovo sve učiniti što i s Blenderom ali naravno na puno teži način. Za učenje ove biblioteke treba jako puno vremena, pogotovo ako ga želimo naučiti kako treba. Three.js je jako velik te ga se ne može naučiti u cijelosti (osim ako ne radimo s njim redovno dugi niz godina). Čak i profesionalni programeri koji redovno koriste Three.js i dalje uče o novim stvarima, prema tome učenje ove biblioteke nikada ne prestaje. Three.js ima odličnu dokumentaciju i jako puno malih gotovih projekata kod kojih možemo vidjeti sav programski kod. Možemo očekivati da ćemo naići na jako puno prepreka, no njih moramo shvatiti kao izazove kojima ćemo postati još bolji. Kada naiđemo na probleme koji nam izgledaju nemoguće za riješiti, dobro je barem pokušati jer se može desiti da postignemo uspjeh, no ako ne uspijemo, moramo se ohrabriti da smo barem pokušali.

Literatura

1. https://threejs.org/
2. https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene
3. https://threejs.org/examples/#webgl_animation_cloth
4. https://threejs-journey.xyz/
5. https://threejs.org/editor/
6. https://github.com/mrdoob/three.js/
7. https://threejsfundamentals.org/
8. https://youtu.be/6oFvqLfRnsU
9. https://www.youtube.com/watch?v=8jP4xpga6yY
10. https://en.wikipedia.org/wiki/Three.js
11. https://redstapler.co/add-3d-model-to-website-threejs/
12. https://davidlyons.dev/threejs-intro/#slide-0
13. https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_on_the_web/Building_up_a_basic_demo_with_Three.js?utm_campaign
14. https://aframe.io/docs/1.2.0/introduction/developing-with-threejs.html
15. https://dev.to/nitinfab/getting-started-with-three-js-2mnf
16. https://www.html5rocks.com/en/tutorials/three/intro/
17. https://hackernoon.com/introduction-to-3d-javascript-library-threejs-basics-aec33yfb
18. http://math.hws.edu/graphicsbook/c5/s1.html
19. https://solutiondesign.com/insights/webgl-and-three-js-lighting/
20. https://x-team.com/blog/3d-webgl-threejs/
21. https://www.creativebloq.com/how-to/get-started-with-webgl-using-threejs
22. https://medium.com/@benjamin.c.coleman/the-beginners-guide-to-beginning-three-js-c36b8947c2aa
23. https://www.smashingmagazine.com/2013/09/introduction-to-polygonal-modeling-and-three-js/
24. https://riptutorial.com/three-js
25. https://www.jesuisundev.com/en/understand-threejs/
26. https://threejs.live/#/
27. https://usersnap.com/blog/3d-graphics-three-js-review/

Slike:

Slika 1 - https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene
Slika 3 - https://threejs.org/docs/index.html?q=box#api/en/geometries/BoxGeometry
Slika 4 - https://threejs.org/docs/index.html?q=text#api/en/geometries/TextGeometry
Slika 5 - https://threejs.org/docs/?q=material#api/en/materials/MeshNormalMaterial
Slika 6 - https://threejs.org/docs/?q=material#api/en/materials/MeshMatcapMaterial
Slika 7 - https://threejs.org/docs/?q=material#api/en/materials/MeshDepthMaterial
Slika 8 - https://threejs.org/docs/?q=material#api/en/materials/MeshLambertMaterial
Slika 9 - https://threejs.org/docs/?q=material#api/en/materials/MeshPhongMaterial
Slika 10 - https://threejs.org/docs/?q=material#api/en/materials/MeshToonMaterial
Slika 11 - https://threejs.org/docs/?q=material#api/en/materials/MeshStandardMaterial
Slika 12 - https://threejs.org/docs/?q=material#api/en/materials/MeshPhysicalMaterial
Slika 13 - https://threejs.org/examples/webgl_materials_envmaps.html
Slika 14 - https://www.intexsoft.com/images/intexsoft/blog/three-js/three-js-4.jpg
Slika 15 - https://threejs.org/editor/
Slika 16 - https://www.intexsoft.com/images/intexsoft/blog/three-js/three-js-3.jpg
Slika 17 - https://threejs.org/examples/#webgl_animation_cloth
Slika 18 - https://www.npmjs.com/package/stats-js

Teksture u projektu:

1. jastuk - https://d2gg9evh47fn9z.cloudfront.net/800px_COLOURBOX10002021.jpg
2. jastuk - https://cdn.hipwallpaper.com/i/78/44/U2HIOx.jpg
3. jastuk - https://cdn.hipwallpaper.com/i/86/93/PbQ2lY.png
4. jastuk - http://supsystic-42d7.kxcdn.com/_assets/forms/img/bg/bg_support_form.jpg
5. jastuk - https://st2.depositphotos.com/3169803/9608/v/950/depositphotos_96085304-stock-illustration-vector-seamless-hi-tech-pattern.jpg
6. jastuk - https://www.artisansguild.co.uk/images/3.png
7. jastuk - https://wallpapercave.com/wp/wp4471366.jpg
8. jastuk - https://wallpapercave.com/wp/wp4471376.jpg

9. jastuk - https://wallpapercave.com/wp/wp4471420.jpg
10. jastuk -
https://www.google.com/url?sa=i&url=https%3A%2F%2Fpngtree.com%2Ffreebackground%2Fblack-tech-sci-fi-design-engine-drawings-hi-tech-future-spaceship-poster_1043681.html&psig=AOvVaw0cAhuXhrvw3LYG771bM7Fe&ust=1621697574963000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCLi71LSM2_ACFQAAAAAdAAAAABAD

Sažetak

Sve do nedavno, nije postojalo puno web stranica koje koriste mogućnosti 3D grafike. Razlozi su razni, no zasigurno najveći razlog su bila ograničenja performansi mobilnih uređaja kao i zahtjevnost pisanja WebGL koda.

U ovome diplomskom radu opisana je glavna razlika između WebGLa i Three.jsa. Pošto je totalnim početnicima dosta teško započeti sa Three.jsom, opisane su tri različite metode uključivanja Three.js biblioteke u projekt. Pokrivene su glavne teme poput: geometrije, materijala, tekstura, svjetla i sjena. Prikazan je način na koji se može iz softvera Blender izvesti 3D model, te isto tako način na koji se može isti uvesti u Three.js projekt. Opisan je način na koji se 3D modeli mogu pomicati, skalirati i rotirati. Osim toga, opisani su i načini korištenja animacija, kamera, kontrola kamera te fizike. Za one koji žele saznati više, objašnjene su teme poput: punog zaslona i promjene veličine zaslona, Debug UIa, te su dodani savjeti za bolje performanse izvođenja web stranice.

Opisana je izrada početne stranice i preostalih stranica u projektu. Osim toga, opisana je izrada, tekstura i izvoz 3D modela u softveru Blender.

Ključne riječi: 3D grafika, WebGL, Three.js, Blender, 3D model

Summary

Until recently, there weren't many websites that took advantage of 3D graphics capabilities. The reasons are various, but certainly the biggest reason was the limitation of mobile device performances as well as the complexity of writing WebGL code.

This thesis describes the main difference between WebGL and Three.js. Since it is quite difficult for absolute beginners to get started with Three.js, three different methods of involving the Three.js library in the project are described. The main topics such as: geometry, materials, textures, light and shadows are covered. The method of exporting 3D objects and importing them to Three.js is described. It is shown how 3D models can be moved, scaled, and rotated. In addition, the ways of using animations, cameras, camera controls and physics are described. For those who want to learn more, topics such as: full screen and screen resizing, Debug UI, and tips for better website performance have been explained.

The creation of the home page and the remaining pages in the project are described. In addition, the production, texture, and export of 3D models in Blender software are described.

Keywords: 3D graphics, WebGL, Three.js, Blender, 3D model