Stvaranje i prikaz 3D grafike u internetskom pregledniku pomoću Three.js biblioteke

Ajder, Nensi

Sveučilište u Rijeci, Fakultet informatike i digitalnih tehnologija

Sveučilišni prijediplomski studij Informatika

Nensi Ajder

Stvaranje i prikaz 3D grafike u internetskom pregledniku pomoću Three.js biblioteke

Završni rad

Mentor: dr. sc. Slobodan Beliga

Rijeka, rujan 2023.

Sveučilište u Rijeci
FIDIT Fakultet informatike i digitalnih tehnologija
www.inf.uniri.hr

Rijeka, 10.5.2023.

Zadatak za završni rad

Pristupnik: Nensi Ajder

Naziv završnog rada: Stvaranje i prikaz 3D grafike u internetskom pregledniku pomoću Three.js biblioteke

Naziv završnog rada na engleskom jeziku: *Creating and displaying 3D graphics in a web browser using the Three.js library*

Sadržaj zadatka: Three.js je JavaScript biblioteka koja se koristi za stvaranje i prikaz 3D računalne grafike u internetskom pregledniku pomoću WebGL-a. Zadatak završnog rada je opisati početke razvoja i trenutno stanje biblioteke Three.js, kao i njezinu namjenu i osnovne funkcionalnosti. Osim teorijskog aspekta, zadatak diplomskog rada je i načiniti praktični primjer koji demonstrira funkcionalnosti Three.js biblioteke u vlastitom primjeru kroz stvaranje i prikaz 3D grafike u internetskom pregledniku. U primjeru je potrebno koristiti jednostavniju ili složenu scenu, različite geometrije i materijale, teksture, svjetla, ali i neke naprednije elemente koji se mogu odnositi na transformacije objekata na sceni, animacije, kamere, kontrole i sl.

Mentor:

dr. sc. Slobodan Beliga

Voditelj za završne radove:

doc. dr. sc. Miran Pobar

Zadatak preuzet: datum

Nensi Ajder

(potpis pristupnika)

Adresa: Radmile Matejčić 2
51000 Rijeka, Hrvatska
Tel: +385(0)51 584 700
E-mail: ured@inf.uniri.hr
OIB: 64218323816
IBAN: HR1524020061400006966
uniri

SAŽETAK

U završnom radu opisano je stvaranje i prikazivanje 3D grafike u internetskom pregledniku pomoću Three.js biblioteke. Rad je organiziran u četiri glavna poglavlja. Osnove ove biblioteke opisane su kroz prvo poglavlje, a obuhvaćaju osnovne definicije, povijest, postavljanje okruženja i izradu prve Three.js aplikacije. Sljedeće poglavlje razrađuje funkcionalnosti Three.js-a kao što su kamera, svjetla, geometrija i materijali i drugo. Treće poglavlje orijentirano je na Three.js dodatke *Orbit Controls* i *dat.GUI* koji korisnicima nude interaktivnost sa Three.js aplikacijom. Zadnje poglavlje biti će posvećeno izradi složenije Three.js aplikacije Sunčevog sustava koja će objediniti sve što je obrađivano u prijašnjim poglavljima. Uz aplikaciju Sunčevog sustava, sve ostale funkcionalnosti biti će potkrepljene primjerima dostupnim na *CD*-u.

Ključne riječi: Three.js, scena, kamera, *renderer*, svjetla, objekti, *loaderi*, *orbit controls*, grafičko korisničko sučelje, Sunčev sustav

SADRŽAJ

UVOD

Kako se potreba za kreiranjem bogatih, dinamičnih i interaktivnih sadržaja na internetskim preglednicima povećava, tako se razvijaju i tehnologije koje bi to omogućile. Jedan od prvih internetskih preglednika koji je omogućavao prikazivanje slika u JPEG i GIF formatima bio je Mosaic. Osnovan je 1993. godine te je njegova mogućnost kombinacije teksta i grafike na istoj stranici označila preokret u budućnosti razvoja internetskih preglednika. Osnovne tehnologije u izradi internetskih stranica poput HTML-a i CSS-a postupno se nadograđuju te su 2010. godine uvedeni HTML5 i CSS3 koji su pružali još veće mogućnosti nad grafikom i stilizacijom. Na razvoj još interaktivnijih, dinamičnijih i vizualno bogatih stranica veliki utjecaj imala je pojava JavaScripta, ta temelju kojeg je kasnije razvijen WebGL (engl. *Web Graphics Library*). WebGL je JavaScript aplikacijsko programsko sučelje (engl. *Application Programming Interface*) koje omogućuje renderiranje 3D grafike u preglednicima.

JavaScript i WebGL doveli su do pojave Three.js-a. Three.js je JavaScript biblioteka temeljena na WebGL-u te je uvelike olakšala kreiranje 3D grafike u preglednicima, pritom omogućavajući i stvaranje puno kompleksnijih i realističnijih scena. Sve o Three.js-u, njegovim funkcionalnostima i dodacima biti će rečeno u idućim poglavljima.

OPĆENITO O THREE.JS BIBLIOTECI

Three.js je JavaScript biblioteka otvorenog koda temeljena na WebGL-u te služi za izradu dinamičnih i interaktivnih internetskih stranica koje koriste dvodimenzionalnu i trodimenzionalnu grafiku. Three.js omogućuje renderiranje grafike izravno u preglednicima. Uz to omogućuje stvaranje složene trodimenzionalne grafike koristeći samo JavaScript, kao i stvaranje scena virtualne i proširene stvarnosti unutar preglednika. Pomoću Three.js-a mogu se dodavati različiti materijali, teksture, animirati trodimenzionalni objekti, učitavati ili raditi s objektima iz drugog softvera za 3D modeliranje. Također, mnogi preglednici ga podržavaju s obzirom da je temeljen na WebGL-u. Za uspješan rad sa Three.js bibliotekom potrebno je poznavati osnove HTML-a, CSS-a i JavaScripta.

HTML (engl. *HyperText Markup Language*) je osnovni jezik za izradu internetskih stranica te daje podatke o njezinoj strukturi i sadržaju, gdje se na temelju unesenih podataka oblikuje stranica kakvu korisnik vidi u internetskom pregledniku. Ukoliko se stranici želi dati estetski privlačniji izgled, koristi se CSS (engl. *Cascading Style Sheets*). CSS surađuje sa HTML dokumentom preko stilova i klasa. JavaScript je visokoprogramski jezik koji daje interaktivnost i dimaničnost internet stranicama, primjerice kod unosa teksta.

POVIJEST

Kao što je već rečeno, Three.js koristi tehnologiju WebGL-a za renderiranje grafike u internetskim preglednicima. WebGL je JavaScript aplikacijsko programsko sučelje (engl. *Application Programming Interface*) koje omogućuje renderiranje 2D i 3D grafike unutar kompatibilnih web preglednika koristeći mogućnosti grafičke procesorske jedinice (engl. *graphics processing unit*). Međutim, WebGL je sustav niske razine koji crta samo osnovne objekte, kao što su točke, kvadrati i linije te je vrlo kompleksan za rad. Kako bi se stvari pojednostavile, Ricardo Cabello 2010. godine osniva jednostavniji i daleko intuitivniji alat za rad sa složenom grafikom za internetske preglednike - Three.js.

INSTALACIJA

Svaki Three.js projekt mora imati najmanje jednu HTML datoteku koja definira web sjedište te JavaScript datoteku koja će pokretati Three.js kod. Načini za implementaciju Three.js-a unutar projekta su raznoliki, a neki od njih uključuju:

1. Preuzimanje cijelog Three.js projekta sa neke stranice, primjerice GitHub-a
2. Korištenje veza sa distribuiranih mreža poslužitelja (engl. *Content Delivery Network*, CDN)
3. Instalacija Three.js paketa pomoću upravitelja paketa čvorova (engl. *Node Package Manager*, NPM)

U radu je korišten pristup bibliotekama pomoću veza na odgovarajuće distribuirane mreže poslužitelja pa će radi toga ovakav princip biti detaljnije razrađen. Stvara se mapa projekta i unutra nje dodaju dvije temeljne datoteke koje će biti potrebno pokrenuti barem preko lokalnog poslužitelja ukoliko se sadržaj želi prikazati na internetskom pregledniku.

Početak je stvaranje dokumenta 'indeks.html' u kojemu je važno stvoriti modul koji će se nadovezivati uz JavaScript datoteku kao što je prikazano na slici 1.

```html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>Prva Three.js aplikacija</title>
        <style>
            body { margin: 0; }
        </style>
    </head>
    <body>
        <script type="module" src="/main.js"></script>
    </body>
</html>
```

Slika 1: Osnovna HTML datoteka

Zatim, na slici 2. prikazan je sljedeći korak. Potrebno je dodati ispravnu CDN vezu te kreirati importnu mapu koja će omogućiti uvoz funkcionalnosti modula Three.js.

```html
<script async
src="https://unpkg.com/es-module-shims@1.3.6/dist/es-module-shims.js"></script
>


<script type="importmap">
  {
    "imports": {
      "three": "https://unpkg.com/three/build/three.module.js"
    }
  }
</script>
```

Slika 2: Dodavanje CDN veze i importnih mapa

Linkovi za dohvaćanje CDN veza:

https://unpkg.com/es-module-shims@1.3.6/dist/es-module-shims.js

https://unpkg.com/three/build/three.module.js

Na kraju se unutar datoteke 'main.js' uvozi Three.js pomoću sljedeće naredbe:

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

PRIMJER OSNOVNE THREE.JS APLIKACIJE

Tri najvažnija koncepta u izradi funkcionalne Three.js aplikacije su scena, kamera i *renderer*. Nakon uvoza Three.js biblioteke definira se scena. Scena je osnovni objekt koji sadržava sve ostale objekte kao što su kamere, svjetla i objekti. Potom se definira i određuje položaj kamere.

Najčešće korišten tip kamere u Three.js-u je perspektivna kamera. Perspektivna kamera daje pogled na 3D scenu po uzoru na ljudski način percepcije. Nakon definirane kamere definira se WebGL *renderer* koji omogućuje prikaz grafike u pregledniku. Definicije najosnovnijih koncepata Three.js-a prikazani su na slici 3.

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

// Definicija scene
const scene = new THREE.Scene();

// Definicija i postavljanje kamere
const camera = new THREE.PerspectiveCamera( 75, window.innerWidth /
window.innerHeight, 0.1, 1000 );
camera.position.z = 5;

// Definicija renderera - WebGLRenderer()
const renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
document.body.appendChild( renderer.domElement );
```

Slika 3: Definicija scene, kamere i renderera

Nakon definirane scene, kamere i *renderera*, potrebno je uključiti neki konkretan objekt u scenu, primjerice kocku. Za kreiranje kocke potrebna je geometrija koja definira oblik te materijal koji definira kako izgleda površina. Nakon toga kombinacijom geometrije i materijala stvara se *mesh*, odnosno objekt koji će biti renderiran i uključen u scenu kao što to prikazuje slika 4.

```
// Definicija objekta (kocke)
const geometry = new THREE.BoxGeometry( 1, 1, 1 );
const material = new THREE.MeshBasicMaterial( { color: 0xbb00ff } );
const cube = new THREE.Mesh( geometry, material );
scene.add( cube );
```

Slika 4: Stvaranje objekta

Još je potrebno definirati funkciju za animaciju objekta. Unutar funkcije `animate` koristi se funkcija `requestAnimationFrame()` koja omogućuje glatku animaciju funkcije koja se prosljeđuje kao parametar, što je u ovom slučaju funkcija `animate()` svaki put kada je ekran spreman za novi okvir (engl. *frame*). U ovom slučaju, svaki puta kada se pozove funkcija `animate`, kocka se rotira malo po x osi, a malo po y osi prostora geometrije te se sukladno s tim renderira scena. Pozivanjem funkcije `animate()` počinje beskonačna petlja koja konstantno rotira kocku i osvježava ekran. Slika 5. prikazuje isječak koda kojom se definira animacija, odnosno rotacija kocke po njezinoj x i y osi.

```
// Definicija funkcije za animaciju objekta 'cube'
function animate() {
    requestAnimationFrame(animate);
    cube.rotation.x += 0.02;
    cube.rotation.y += 0.02;
    renderer.render( scene, camera );
}

animate();
```

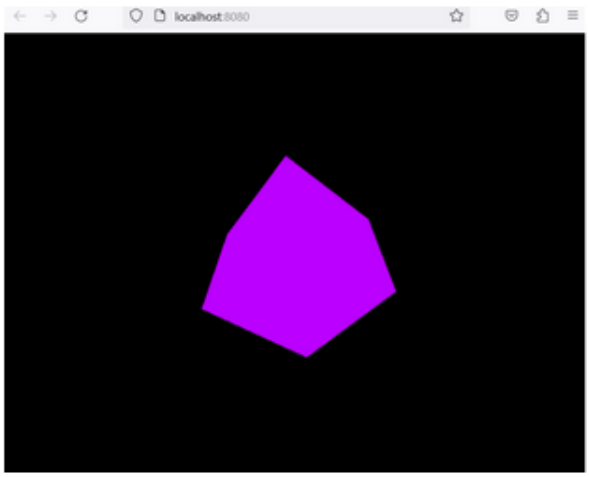
Slika 5: Funkcija animacije i rotacije kocke

Na kraju je u svakoj Three.js aplikaciji poželjno implementirati osluškivač (engl. *listener*) koji prati situaciju o prozoru na kojemu se izvodi aplikacija, odnosno njegovoj širini i visini. Varijable `width` i `height` dohvaćaju trenutne podatke o veličini prozora. Potom funkcionalnost `renderer.setSize(width, height)` omogućuje postavljanje veličine renderera na novu vrijednost te tako osigurava da će 3D prikaz u Three.js-u uvijek ispuniti cijeli prostor, neovisno o njegovoj veličini. Funkcionalnost `camera.aspect` omogućuje ažuriranje omjera širine i visine kako bi 3D prikaz izgledao ispravno te `camera.updateProjectionMatrix()` ažurira projekcijsku matricu kamere i tako osigurava ispravnu projekciju i prikaz 3D scene s obzirom na novi omjer širine i visine. Slika 6. prikazuje funkciju osluškivača koji prilagođava scenu s obzirom na omjer širine i visine prozora.

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

Slika 6: Osluškivač za prilagođavanje scene po omjeru prozora

Time je kreirana funkcionalna aplikacija koristeći Three.js te se njezinim pokretanjem dobiva ljubičasta kocka koja se rotira istovremeno po x i y osi kao što je prikazano na slici 7.



Slika 7: Prikaz rotirajuće kocke u internetskom pregledniku

FUNKCIONALNOSTI THREE.JS BIBLIOTEKE

Three.js pruža veliki broj mogućnosti te će u ovom poglavlju biti objašnjeno korištenje ključnih elemenata i funkcionalnosti kao što su kamera, geometrija i materijali, svjetla, grupiranja, učitavanje modela, teksta i tekstura.

KAMERA

Kamera je element koji "snima" trodimenzionalnu scenu te je transformira u dvodimenzionalnu sliku koja se potom prikazuje na ekranu. Postoje razne vrste kamera, ali glavne su perspektivna kamera i ortogonalna kamera. Perspektivna kamera (engl. *Perspective Camera*) stvara osjećaj dubine te simulira način na koji ljudsko oko vidi stvari, odnosno bliži objekti se čine većima, a dalji manjima. S druge strane, ortogonalna kamera (engl. *Ortographic Camera*) nema perspektivnog izobličenja te bez obzira na udaljenost kamere objekti zauzimaju istu količinu prostora na ekranu.

PARAMETRI I PRIMJENA PERSPEKTIVNE KAMERE

Parametri konstruktora perspektivne kamere *`PerspectiveCamera( fov : Number, aspect : Number, near : Number, far : Number )`* označavaju sljedeće:

- 'fov' (*Field of View*): Određuje koliko će scena biti široka. Manje vrijednosti daju uži pregled scene, a veće vrijednosti širi pregled.
- 'aspect' (*Aspect Ratio*): Određuje se odnos širine ekrana prema njegovoj visini.
- 'near' (*Near Clipping Plane*): Određuje najbližu točku koja će biti vidljiva kameri, odnosno sve što je bliže od te vrijednosti neće biti renderirano.
- 'far' (*Far Clipping Plane*): Određuje najdalju točku koja će biti vidljiva kameri te sve što je dalje od te vrijednosti neće biti renderirano.

Sljedeći kod prikazuje inicijalizaciju perspektivne kamere:

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

scene.add(camera);
```

Perspektivna kamera se koristi za izradu realističnih 3D aplikacija, kao što su video igre i simulacije.

PARAMETRI I PRIMJENA ORTOGONALNE KAMERE

Parametri unutar konstruktora ortogonalne kamere *`OrthographicCamera( left : Number, right : Number, top : Number, bottom : Number, near : Number, far : Number )`* definiraju lijevu, desnu, gornju i donju granicu vidnog polja kamere, a uz to poprima parametre 'near' i 'far' koji imaju isto značenje kao i kod perspektivne kamere.

Sljedeći kod prikazuje inicijalizaciju ortogonalne kamere:

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

```
scene.add( camera );
```

Ortogonalna kamera prikazuje objekte bez perspektivne distorzije, pa se ona koristi u tehničkim crtežima, arhitektonskim planovima ili nekim određenim vrstama video igara.

SVJETLO

Svjetlo u Three.js-u predstavlja jedan od ključnih elemenata za kreiranje realistične trodimenzionalne scene. Postoje razni tipovi osvjetljenja koji se koriste zavisno o tome kakav se efekt želi postići. Najčešće korišteni tipovi osvjetljenja su ambijentalno svjetlo (engl. *Ambient light*), usmjereno svjetlo (engl. *Directional light*) te točkasto svjetlo (engl. *Point light*). Treba napomenuti da je prilikom uključivanja osvjetljenja u scenu važno imati na umu da neke vrste materijala kao što su *Mesh Basic Material* ne reagiraju na osvjetljenje, dok drugi kao što su *Mesh Standard Material* ili *Mesh Physical Material* reagiraju. O materijalima će biti riječ u sljedećem poglavlju.

AMBIJENTALNO SVJETLO

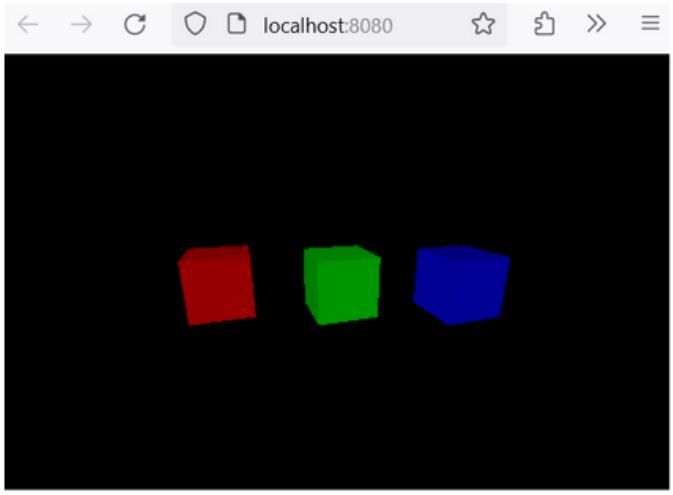
Osvjetljuje čitavu scenu podjednako te se svjetlost rasprostire u svim smjerovima i udaljenostima. Jednako utječe na sve osvjetljene objekte u sceni te dodaje boju materijalu objekta. Konstruktor `AmbientLight( color : Integer, intensity : Float )` kao parametre prima boju koja je numerička vrijednost RGB sustava boja te intenzitet koji govori o jačini osvjetljenja.

Sljedeći kod prikazuje inicijalizaciju ambijentalnog svjetla

```
const ambientLight = new THREE.AmbientLight(0xffffff);

scene.add(ambientLight);
```

Primjer kako takva vrsta osvjetljenja djeluje na objekte prikazano je na Slici 11.



Slika 8: Ambijentalno svjetlo

USMJERENO SVJETLO

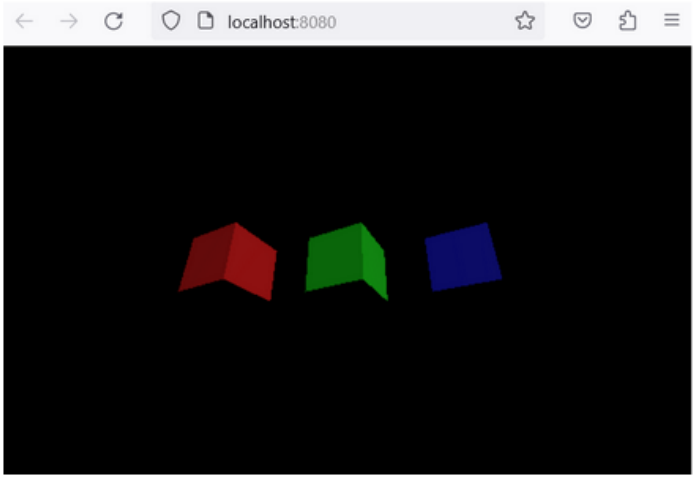
Usmjereno svjetlo ima svoj izvor, odnosno dolazi iz određene točke te se emitira izravno prema cilju. Sve svjetlosne zrake koje dolaze iz tog određenog izvora su paralelne, a najbolji primjer takvog tipa osvjetljenja je Sunce. Konstruktor *`DirectionalLight( color : Integer, intensity : Float )`* prima iste parametre kao i ambijentalno svjetlo, ali je potrebno dodati njegovu poziciju, što je prikazano u kodu:

```
const directionalLight = new THREE.DirectionalLight(0xffffff, 1);

directionalLight.position.set(1, 1, 1).normalize();

scene.add(directionalLight);
```

Slika 13. prikazuje djelovanje usmjerenog svjetla na objekte te je time vidljiva razlika između takvog i ambijentalnog. Dok ambijentalno svjetlo osvjetljuje svaki dio scene podjednako, usmjereno svjetlo djeluje na ostale objekte s obzirom na poziciju svjetla.



Slika 9: Usmjereno svjetlo

TOČKASTO SVJETLO

Ovakav tip osvjetljenja u određenoj poziciji stvara izvor svjetlosti koji se nadalje širi u svim smjerovima. Parametri konstruktora *`PointLight( color : Integer, intensity : Float, distance : Number, decay : Float )`* označavaju:

- '*color*': boja svjetlosti
- '*intensity*': jačina svjetla
- '*distance*': maksimalni doseg svjetlosti
- '*decay*': stopa slabljenja osvjetljenja sa udaljenosti od izvora

Znači, najprije se inicijalizira točkasto svjetlo, a potom se definira na kojoj poziciji će se ona nalaziti slično kao i kod usmjerenog svjetla te njezini parametri.

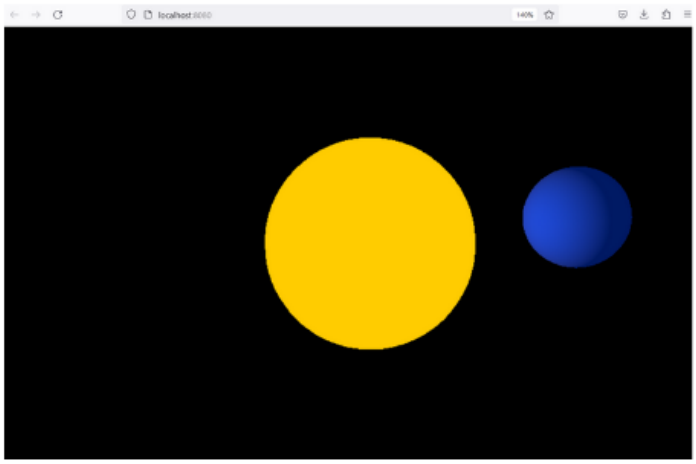
Sljedeći kod prikazuje inicijalizaciju točkastog svjetla i postavljanje istog na određenu poziciju:

```
const pointLight = new THREE.PointLight(0xffffff, 500, 0);

pointLight.position.set(1, 0, 0);
```

```
scene.add(pointLight);
```

Slika 15. prikazuje način upotrebljavanja točkastog svjetla koje se nalazi unutar sfere kako bi ona prividno služila kao izvor svjetlosti za drugu sferu. Pozicija sfere koja je "izvor svjetlosti" je ista kao i pozicija svjetlosnog izvora. Vidljivo je kako je plavo obojana sfera svjetlija dijelom okrenutim prema žutoj sferi.



Slika 10: Točkasto svjetlo

GEOMETRIJA I MATERIJALI

Geometrija (engl. *Geometry*) se koristi za kreiranje i definiranje oblika u Three.js-u. Three.js nudi mnoštvo različitih preddefiniranih geometrija. To mogu biti jednostavni oblici kao što su kocka ili sfera, a mogu biti i složeniji oblici kao što su dodekaedar ili torus. Nadalje, važan pojam u geometriji je materijal (eng. *Material*). Materijal služi za definiranje boje ili teksture geometrije. Kao što je spomenuto u prijašnjem poglavlju, postoje razni tipovi materijala kao što su *Mesh Basic Material*, *Mesh Phong Material* ili *Mesh Toon Material*.

Međusobno se razlikuju po svojstvima kao što su sjaj i djelovanje na svjetlo/sjenu. Treća važna komponenta je *mesh* koja kombinirajući geometriju i materijal stvara objekt koji se na kraju dodaje sceni. U ovom poglavlju će biti detaljnije objašnjene spomenute geometrije i materijali.

GEOMETRIJA PRAVOKUTNIKA

U Three.js-u *Box Geometry* kao i svaki pravokutnik ima obilježja kao što su visina (engl. *height*), širina (engl. *width*) i dubina (engl. *depth*) pa je tako konstruktor oblika `BoxGeometry(width : Float, height : Float, depth : Float, widthSegments : Integer, heightSegments : Integer, depthSegments : Integer).`

Sljedeći dio koda prikazuje inicijalizaciju geometrije pravokutnika, kao i njegovo uključivanje na scenu:

```
const geometry = new THREE.BoxGeometry( 1, 1, 1 );

const material = new THREE.MeshBasicMaterial( {color: 0xffff00} );

const cube = new THREE.Mesh( geometry, material );
```

```
scene.add( cube );
```

GEOMETRIJA SFERE

Sphere Geometry omogućuje kreiranje sfere. Konstruktor *`SphereGeometry(radius : Float, widthSegments : Integer, heightSegments : Integer, phiStart : Float, phiLength : Float, thetaStart : Float, thetaLength : Float)`* prima veliki broj parametara:

- '*radius'*: polumjer sfere
- '*widthSegments'*: broj horizontalnih segmenata
- '*heightSegments'*: broj vertikalnih segmenata
- '*phiStart'*: horizontalni početni kut iz kojeg sfera počinje
- '*phiLength'*: horizontalni kut kroz koji se sfera proteže
- '*thetaStart'*: vertikalni početni kut iz kojeg sfera počinje
- '*thetaLength'*: vertikalni kut kroz koji se sfera proteže

Međutim, u praksi se uglavnom definiraju prva tri parametra kao što je prikazano u isječku koda:

```
const geometry = new THREE.SphereGeometry( 15, 32, 16 );
const material = new THREE.MeshBasicMaterial( { color: 0xffff00 } );
const sphere = new THREE.Mesh( geometry, material ); scene.add( sphere
);
```

GEOMETRIJA DODEKAEDRA

Dodekaedar je geometrijsko tijelo omeđeno s dvanaest ploha koje imaju oblik peterokuta. U Three.js-u postoji predefinirana geometrija za ovakvo geometrijsko tijelo. Konstruktor za dodekaedar *`DodecahedronGeometry(radius : Float, detail : Integer)`* prima samo dva parametra, polumjer (engl. *radius*) i detalj. Parametar *detail* ima zadanu vrijednost 0, a mijenjanjem te vrijednosti na neki broj veći od 0 dodaje vrhove te samim time više nije riječ o dodekaedru. Jednostavan kod za kreiranje dodekaedra prikazano je na slici 18.

```
const geometry = new THREE.DodecahedronGeometry( 0.5 );

const material = new THREE.MeshBasicMaterial( { color: 0xffff00 } );

const dodecahedron = new THREE.Mesh( geometry, material ); scene.add(
dodecahedron );
```

GEOMETRIJA TORUSA

Torus je geometrijsko tijelo koje ima oblik prstena ili krafne. Konstruktor *`TorusGeometry(radius : Float, tube : Float, radialSegments : Integer, tubularSegments : Integer, arc : Float)`* označava sljedeće:

- '*radius'*: polumjer torusa
- '*tube'*: polumjer cijevi torusa

- ‘*radialSegments’* i ‘*tubularSegments’* imaju sličnu ulogu kao i kod sfere
- ‘*arc’*: središnji kut

Sljedeći kod prikazuje inicijalizaciju geometrija torusa i njegovo stvaranje:

```
const geometry = new THREE.TorusGeometry( 10, 3, 16, 100 );

const material = new THREE.MeshBasicMaterial( { color: 0xffff00 } );

const torus = new THREE.Mesh( geometry, material ); scene.add( torus
);
```

MESH BASIC MATERIAL

Mesh Basic Material je osnovni materijal u Three.js-u. Ovaj materijal je samoosvjetljen te ostala rasvjeta nema utjecaj na njega. S obzirom da je ovaj materijal otporan na djelovanje svjetla i sjene, ponekad je teško razlikovati dvije susjedne površine iste boje. Taj problem je naročito vidljiv kod sfere, koja će u tom slučaju izgledati kao 2D krug, a zapravo će biti 3D. Konstruktor `MeshBasicMaterial( parameters : Object )` može poprimati razne parametre, gdje je jedan od najčešće korištenih parametar *color* koji definira boju geometrije. Još jedan koristan parametar je *wireframe* koji prikazuje materijal kao žičani model.

MESH PHONG MATERIAL

Mesh Phong Material je materijal koji simulira sjajne površine s refleksivnim osvjetljenjima. Konstruktor `MeshPhongMaterial( parameters : Object )` prima razne parametre. Osim parametra za definiranje boje geometrije *color.* drugi važni parametri za ovaj tip materijala su *emissive* koji omogućuje da boja simulira svjetlosnu emisiju iz materijala te *shininess* koji određuje koliko je površina sjajna.

MESH STANDARD MATERIAL

Ovakav tip materijala daje realističniji prikaz nego li primjerice ranije spomenuti *Mesh Phong Material* te je glavna ideja stvoriti materijal koji će dati “točan” prikaz u svim scenarijima osvjetljenja. Neki od parametara koji se često koriste unutar konstruktora `MeshStandardMaterial( parameters : Object )` su:

- ‘*color’*: osnovna boja materijala
- ‘*roughness’*: daje grubost površini
- ‘*metalness’*: daje izgled metalika

MESH TOON MATERIAL

Mesh Toon Material je vrsta materijala koja služi za postizanje crtanog efekta. Iako su svi prijašnje spomenuti parametri važni i za ovaj tip materijala, treba napomenuti da postoji mogućnost umetanja gradijentne mape koja definira kako se sjenčanje mijenja prema kutu između površine i svjetlosnog izvora. Nadalje se koristi parametar ‘gradientMap’ unutar konstruktora `MeshToonMaterial( parameters : Object )` i tako omogućava bolji crtani efekt.

PRIMJER THREE.JS APLIKACIJE NA TEMU GEOMETRIJE I MATERIJALA

U primjeru je vidljivo korištenje četiri geometrija, a svaka od njih sastoji se od drugog tipa materijala. Objekti su poredani u stilu 2x2 rešetke (engl. *grid*) te će se iz tog razloga upotrebljavati malo drugačiji pristup njima, odnosno biti će korištena dvostruka for petlja.

Kao i kod svake Three.js aplikacije potrebno je postaviti temelj aplikacije, odnosno scenu, kameru i *renderer*. S obzirom da se radi o demonstraciji geometrije i materijala, važno je postaviti osvjetljenje. U primjeru postavljena su dva osvjetljenja, ambijentalno i usmjereno. S obzirom da se ne zahtijevaju animacije specifičnih objekata već je potrebno samo renderirati i prikazati sadržaj na web pregledniku, moguće je odmah dodati funkciju `animate()` čime je završen prvi dio aplikacije te je nadalje moguće dodavati objekte koji će biti vidljivi.

Geometrije i materijali definirani su kao liste te obuhvaćaju one geometrije (pravokutnik, sfera, dodekaedar i torus) i materijale (*Mesh Basic Material*, *Mesh Phong Material*, *Mesh Standard Material*, *Mesh Toon Material*) koji su bili obrađeni u ovom poglavlju. Kao parametri materijala također su korišteni oni obrađeni. Slika 11. prikazuje pripadajući kod definicija geometrija i materijala.

```
// Definiranje geometrija
const geometries = [
    new THREE.BoxGeometry(1, 1, 1),
    new THREE.SphereGeometry(0.5, 32, 32),
    new THREE.DodecahedronGeometry(0.5),
    new THREE.TorusGeometry(0.5, 0.2, 16, 100)
];

// Definiranje materijala
const materials = [
    new THREE.MeshBasicMaterial({
        color: 0xaa00aa,
        wireframe: true
    }),
    new THREE.MeshPhongMaterial({
        color: 0xaa0000,
        emissive: 0x000022,
        shininess: 30
    }),
    new THREE.MeshStandardMaterial({
        color: 0xaaff66,
        roughness: 0.2,
        metalness: 0.5
    }),
    new THREE.MeshToonMaterial({
        color: 0xffaa00,
    })
];
```

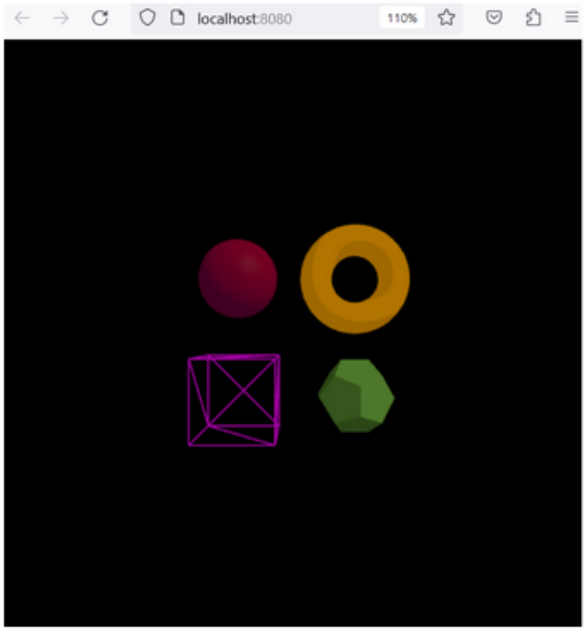
Slika 11: Definicija i stvaranje raznih geometrija i materijala

Nakon definiranja nizova geometrija i materijala potrebno je definirati dvije varijable, `spaceAmount` i `index`. Varijabla `spaceAmount` definira slobodan prostor među objektima, a `index` služi za praćenje položaja unutar nizova `geometries` i `materials`. Potom će se korištenjem dvostruke for petlje generirati četiri različita objekta različitih materijala na sceni, pritom tvoreći 2x2 rešetku, a način realizacije prikazuje slika 12.

```
// Petlja za generiranje objekata (mesheva) i definiranje njihovih pozicija
for (let i = 0; i < 2; i++) {
    for (let j = 0; j < 2; j++) {
        const object = new THREE.Mesh(geometries[index], materials[index]);
        object.position.x = (i * (1 + spaceAmount)) - 0.5 - (spaceAmount / 2);
        object.position.y = (j * (1 + spaceAmount)) - 0.5 - (spaceAmount / 2);
        scene.add(object);
        index++;
    }
}
```

Slika 12: Generiranje četiri objekata po principu 2x2 rešetke

Vanjska for petlja prolazi kroz dva reda objekata, dok unutarnja for petlja prolazi kroz dvije kolone objekata za svaki red. Unutar petlji stvara se objekt koristeći trenutku geometriju i materijal na temelju trenutnog indeksa. Postavlja se pozicija na x osi trenutnog objekta. Uzima se trenutna vrijednost `i`, množi sa `1 + spaceAmount` kako bi se dodao razmak te se zatim oduzima 0.5 i polovica `spaceAmount` varijable kako bi se objekti postavili u središte. Nakon toga se na sličan princip objekt postavlja na poziciju y osi. Dodaje se trenutno kreirani objekt na sceni te se na kraju `index` povećava za 1 kako bi se u sljedećoj iteraciji stvorio novi objekt. Slika 13. prikazuje aplikaciju s četiri kreirana objekta različitih materijala.



Slika 13: Prikaz raznih geometrija i materijala

GRUPIRANJE

Mogućnost grupiranja objekata važna je značajka Three.js-a i može poprilično olakšati i uštediti vrijeme. Grupiranje je moguće izvršiti po principu ‘dijete-roditelj’. Objekt ‘dijete’ metodom `add` dodajemo objektu ‘roditelj’. Primjerice:

```
objektRoditelj.add(scene);
objektDijete.add(objektRoditelj);
```

Grupiranje je moguće vršiti i način da se naprave grupe korištenjem klase `THREE.Group` na način:

```
const group = new THREE.Group();
```

Te se nakon stvaranja dodaju željeni objekti unutar grupe:

```
group.add(object1);
group.add(object2);
...
```

UČITAVANJE MODELA I TEKSTA

Three.js nudi mogućnost umetanja modela, fonta (teksta) i tekstura kako bi se kreirale složenije trodimenzionalne scene. Potrebno je koristiti formate koje Three.js podržava, a to su primjerice '.gltf', '.obj', 'fbx' (za modele) ili '.json' (za fontove). U svrhu učitavanja modela, fonta i tekstura koriste se *loaderi*. Jedni od najvažnijih *loadera* su *GLTF Loader* (za učitavanje modela), *Font Loader* (za učitavanje fonta) i *Texture Loader* (za učitavanje tekstura). Važna napomena prilikom uvoza ovih funkcionalnosti u Three.js projekt ukoliko se koristi CDN jest ta da se umjesto importnih mapa automatski u JavaScript datoteku dodaje import *loadera* preko odgovarajuće putanje, odnosno:

https://unpkg.com/three@<verzija>/examples/jsm/loaders/<ImeLoadera>.js

GLTF *LOADER*

GLTF *Loader* (eng. *Graphics Library Transmission Format*) omogućuje učitavanje modela u JSON (.gltf) ili binarnom (.glb) formatu. Inicijalizacija i učitavanje modela izvodi se kao što je prikazano na sljedećem isječku koda:

```
// UČITAVANJE LOADER INSTANCE

const loader = new THREE.GLTFLoader();

// UČITAVANJE GLTF MODELA

loader.load('model.gltf', (gltf) => {

    const model = gltf.scene;

    scene.add(model);

});
```

FONT *LOADER*

Ovakav *loader* omogućuje učitavanje fonta JSON formata. Nakon definiranog fonta potrebno je izgraditi tekstualni objekt, što se izvršava pomoću posebne vrste geometrije teksta (engl. *Text*

Geometry). Treba napomenuti da je geometriju teksta potrebno posebno *importati* koristeći se putanjom:

https://unpkg.com/three@0.156.1/examples/jsm/geometries/TextGeometry.js

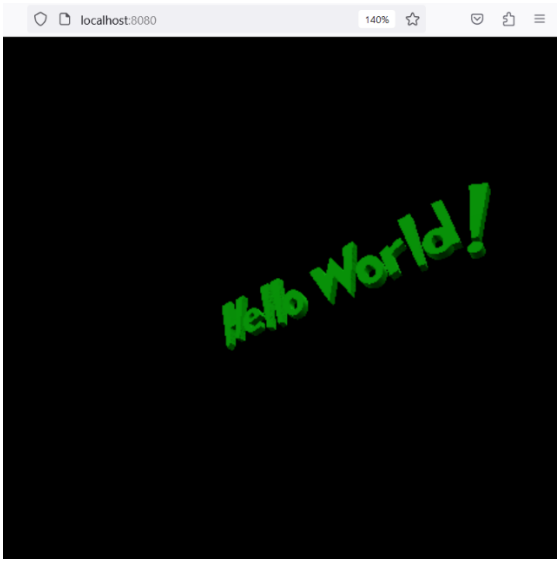
Konstruktor 'TextGeometry(text : String, parameters : Object)' može primati razne parametre, kao što su:

- '*font*': font koji se koristi
- '*size*': veličina teksta
- '*height*': debljina teksta

Primjer inicijalizacije font *loadera*, učitavanje fonta te kreiranje geometrije teksta prikazano je na sljedećem isječku koda:

```
// INICIJALIZACIJA LOADER INSTANCE
const loader = new THREE.FontLoader();
// UČITAVANJE FONTA
loader.load('font.json', (font) => {
    // KREIRANJE TEKSTUALNOG OBJEKTA
    const textGeometry = new THREE.TextGeometry('Hello, Three.js!', {
        font: font,
        size: 0.5,
        height: 0.1,
    });
    const textMaterial = new THREE.MeshBasicMaterial({ color: 0x00ff00
});
    const textMesh = new THREE.Mesh(textGeometry, textMaterial);
    scene.add(textMesh);
});
```

Ukoliko se sve ispravno napravi, internetski preglednik trebao bi prikazati isto što je vidljivo na slici 14.



Slika 14: "Hello World!" u Three.js-u

TEXTURE LOADER

Texture Loader je korišten za učitavanje slika koje se nadalje mogu primijeniti na materijale kako bi se postiglo kreiranje realističnijih objekata. Primjer inicijalizacije *Texture Loadera,* učitavanje slike te primjena iste na materijal je prikazan na isječku koda:

```
// INICIJALIZACIJA LOADER INSTANCE
const loader = new THREE.TextureLoader();
// UČITAVANJE TEKSTURE
loader.load('texture.jpg', (texture) => {
    const material = new THREE.MeshBasicMaterial({ map: texture });
    const geometry = new THREE.BoxGeometry();
    const cube = new THREE.Mesh(geometry, material);
    scene.add(cube);
});
```

DODACI ZA THREE.JS

Uz ključne funkcionalnosti spomenute u prijašnjem poglavlju, Three.js nudi još puno dodatnih funkcionalnosti koje mogu omogućiti korisniku interaktivnost, odnosno suradnju i manipulaciju nad Three.js scenom. Dodaci koji će biti obrađeni u poglavlju su *Orbit Controls* koji služi kako bi se korisnik mogao kretati scenom, te *dat.GUI* koji je zapravo grafičko korisničko sučelje, odnosno korisnik automatski u internetskom pregledniku može promijeniti neka odabrana svojstva bez potrebe mijenjanja istih u kodu.

ORBIT CONTROLS

Orbit Controls omogućuje korisnicima upravljanje kamerom, odnosno rotaciju, pomicanje i zumiranje kamere pomoću uređaja kao što su miš kako bi istražili 3D prostor. Kada se radi s tim dodatkom, prvo ga je potrebno uključiti unutar 'index.html' i 'main.js' datoteka te će isto biti izvedeno putem CDN veza. U HTML datoteci dodaje se `three/controls` u importnu mapu te njemu odgovarajuća putanja do izvora funkcionalnosti dodatka 'OrbitControls.js' što je prikazano na slici 15.

```
"three/controls":
"https://unpkg.com/three/examples/jsm/controls/OrbitControls.js"
```

Slika 15: Umetanje veze za Orbit Controls unutar importne mape u HTML dokumentu

Link za dohvat dodatka *Orbit Controls*:

https://unpkg.com/three/examples/jsm/controls/OrbitControls.js

Nadalje, u JavaScript datoteci se uključuje dodatak te se inicijaliziraju kontrole kao što je prikazano u dijelu koda:

```
import { OrbitControls } from 'three/controls';

const controls = new OrbitControls(camera, renderer.domElement);
```

Ovakav konstruktor 'OrbitControls(object : Camera, domElement : HTMLDOMElement)' prima objekt koji će u pravilu biti kamera kojom se želi manipulirati te drugi parametar 'domElement' koji predstavlja HTML element na kojem će se kontrola događati, a to je u pravilu *renderer*.

Kako bi kontrole uspješno funkcionirale, potrebno je unutar funkcije za animaciju iskoristiti funkciju `update()` na definirane kontrole kako bi se one redovito ažurirale na način:

```
// ANIMACIJA

function animate() {

    requestAnimationFrame(animate);

    controls.update();

    renderer.render(scene, camera);

}
```

Ovime je uspješno postavljen temelj Three.js aplikacije koja koristi dodatak *Orbit Controls*.

Kako bi se svojstva kontroliranja kamerom podesila prema željama korisnika, ovaj dodatak nudi mnoštvo opcija te su neke od značajnijih:

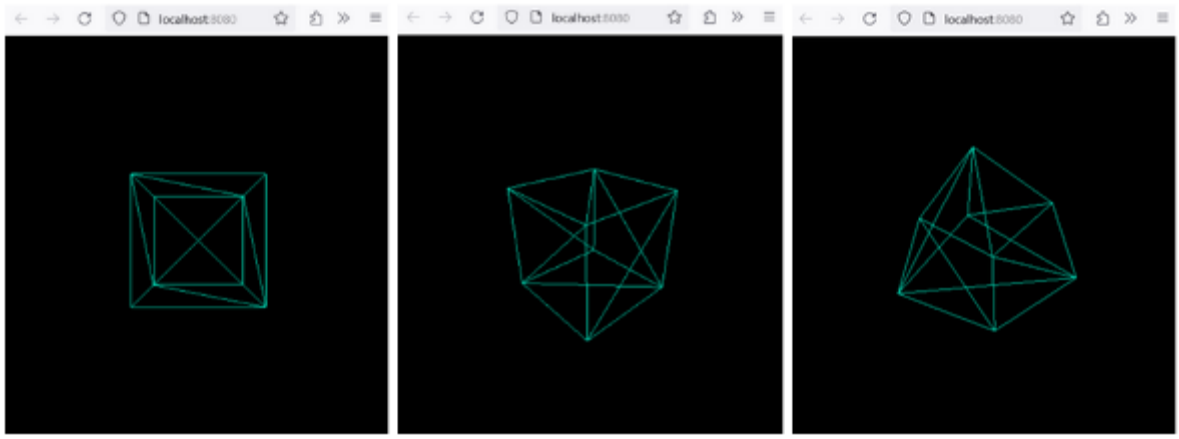
- '*enableZoom'*: omogućuje/onemogućuje zumiranje kamerom
- '*enableRotate'*: omogućuje/onemoguće rotaciju kamere
- '*autoRotate'*: postavljanjem ove vrijednosti na *true* pokreće se automatska rotacija kamere
- '*autoRotateSpeed'*: ukoliko je '*autoRotate'* aktiviran, ova opcija postavlja brzinu automatske rotacije kamere
- '*enableDamping'*: omogućuje glatko usporavanje kretanja kamere, čime se njezino kretanje čini prirodnijim
- '*dampingFactor'*: ukoliko je '*enableDamping'* aktiviran, ova opcija kontrolira brzinu usporavanja kamere

Primjerice, u sljedećem isječku prikazano je dodavanje efekta glatkog usporavanje kretanja kamere.

```
controls.enableDamping = true;

controls.dampingFactor = 0.05;
```

Slika 16. prikazuje interakciju korisnika s Three.js aplikacijom koja koristi dodatak *Orbit Controls*. Treba napomenuti da u ovom slučaju nije kocka ta koja se rotira, već kamera putuje scenom dajući takav dojam.



Slika 16: Manipulacija kamerom putem Orbit Controls dodatka

DAT.GUI

U JavaScript-u, *dat.GUI* je jednostavno grafičko korisničko sučelje (engl. *Graphical user interface*, GUI) koje omogućuje manipulaciju varijablama različitih tipova. U Three.js-u ovaj dodatak se koristi kako bi se mogle izvesti promjene na sceni, primjerice neka svojstva geometrije, materijala ili kamere bez potrebe diranja koda. U poglavlju će biti obrađeno postavljanje ovog dodatka te objašnjena izrada Three.js aplikacije koja ga koristi.

Potrebno je povezati grafičko korisničko sučelje sa projektom, pa se u HTML datoteci unutar `<script></script>` nadodaje sljedeći link:

https://cdnjs.cloudflare.com/ajax/libs/dat-gui/0.7.9/dat.gui.min.js

Instalacija je moguća i koristeći upravitelj paketa čvorova, odnosno NPM-a putem naredbe: *npm install --save dat.gui*.

Nakon toga GUI se unutar Three.js-a definira na način:

```
Const gui = new dat.GUI();
console.log(gui)   // ukoliko se koristi CDN
```

Nakon uspješnog postavljanja grafičkog korisničkog sučelja, potrebno je stvoriti atribute geometrije koje se želi izmijeniti. Ti atributi varirati će ovisno o tome o kojoj geometriji je riječ. Način definiranja istog biti će objašnjeno malo kasnije.

Metodom `add` u grafičko korisničko sučelje dodaju se sljedeći parametri:

- *'object'*: varijabla koju korisnik promatra
- '*property'*: svojstvo koje je potrebno promijeniti
- *'[min]'*: minimalna vrijednost koje GUI pokazuje
- *'[max]'*: maksimalna vrijednost koje GUI pokazuje
- *'[step]'*: stopa smanjivanja ili povećavanja

PRIMJER THREE.JS APLIKACIJE KOJA KORISTI *DAT.GUI*

Ovaj primjer prikazuje toruski čvor čija određena svojstva korisnik može modificirati (radijus i cijev) preko grafičkog korisničkog sučelja *dat.gui*. Umetanje grafičkog korisničkog sučelja i njegova inicijalizacija napravljena je po uzoru na prethodno potpoglavlje.

Aplikacija koristi definiran toruski čvor kao što je vidljivo na Slici 17.

```
const geometry = new THREE.TorusKnotGeometry( 10, 3, 100, 16 );
const material = new THREE.MeshPhongMaterial({
    color: 0x3311aa,
    emissive: 0x072534,
    specular: 0x1d1d1d,
    shininess: 100,
});
const torusKnot = new THREE.Mesh( geometry, material );
```

Slika 17: Definicija toruskog čvora

Grafičko korisničko sučelje potrebno je postaviti na način da se mogu izmjenjivati dva svojstva ove geometrije, a to su *radius* i *tube*. U tu svrhu definiraju se atributi po točnim nazivima svojstava određene geometrije na način prikazan u kodu:

```
const params = {
    radius: 10,
    tube: 3,
};
```

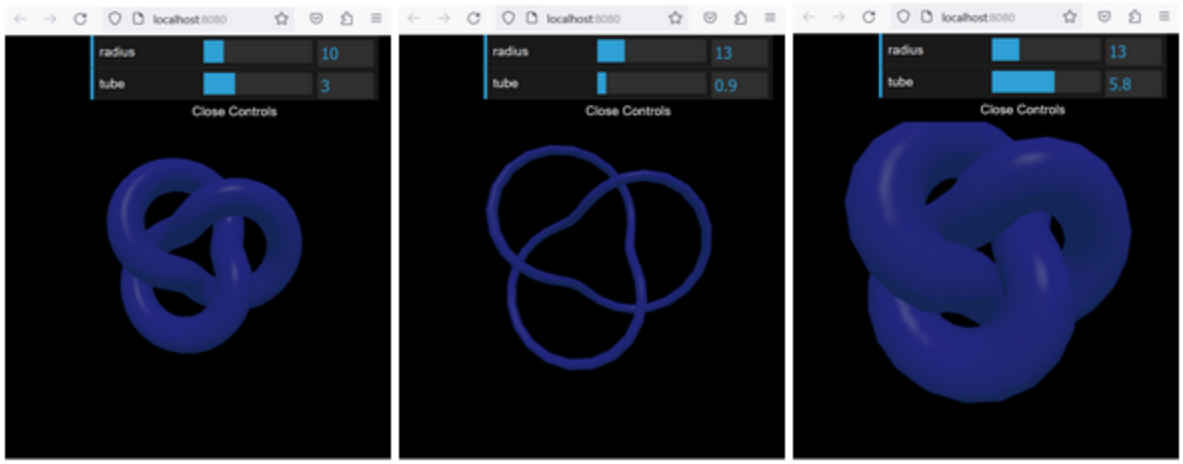
Nakon toga metodom `add()` se definiraju svojstva, nazivi i pozivi funkcija:

```
gui.add(params, 'radius', 1, 50).onChange(function (value) {
    updateGeometry();
});
gui.add(params, 'tube', 0.1, 10).onChange(function (value) {
    updateGeometry();
});
```

Gdje prvi parametar dohvaća listu s atributima, drugi parametar dohvaća određeno svojstvo iz te liste, treći parametar definira najmanju brojčanu vrijednost na koju će biti moguće postaviti svojstvo, dok četvrti parametar definira najveću brojčanu vrijednost svojstva. Definira se rukovatelj događajima (engl. *Event handler*) `onChange(function (value) {})` koji će detektirati promjene te pozvati funkciju `updateGeometry()` koja će se biti pozvana svaki puta kada rukovatelj događajima detektira promjenu. Funkcija `updateGeometry()` potom uništava geometriju objekta koja je postojala do trenutka promjene te stvara novu geometriju sa novim parametrima u isječku koda:

```
function updateGeometry() {
    torusKnot.geometry.dispose();
    torusKnot.geometry = new THREE.TorusKnotGeometry(params.radius, params.tube);
}
```

Na kraju, Slika 18. prikazuje funkcionalnost ovakve aplikacije, odnosno mogućnost korisnika da modificira odabrane parametre geometrije objekta.



Slika 18: Manipulacija geometrije toruskog čvora putem GUI-a

IZRADA SUNČEVOG SUSTAVA U THREE.JS-U

U ovom poglavlju izrađujem Sunčev sustav koristeći biblioteku Three.js. Cilj je da taj projekt obuhvati ono o čemu je bila riječ u prijašnjim poglavljima te da pokaže funkcionalnosti Three.js-a u malo drugačijem pogledu, odnosno da on može biti daleko kompleksniji ukoliko se svi elementi pravilno kombiniraju.

U zadatku je korištena perspektivna kamera, ambijentalno i točkasto svjetlo, geometrije sfere i prstena, materijali *Mesh Basic Material* i *Mesh Standard Material*, *Font Loader*, dodaci *Orbit Controls* i *dat.GUI* te animacije koje omogućuju rotaciju prostora i planeta oko osi, kao i njihovu mogućnost da zaobilaze Sunce.

Naredbom import uvedene su sve biblioteke i dodaci potrebni za kreiranje projektnog zadatka kao što je prikazano na slici 19.

```
import * as THREE from 'three';
import { OrbitControls } from 'three/controls';
import { FontLoader } from 'https://unpkg.com/three@0.156.1/examples/jsm/loaders/FontLoader.js';
import { TextGeometry } from 'https://unpkg.com/three@0.156.1/examples/jsm/geometries/TextGeometry.js';
```

Slika 19: Three.js, Orbit Controls, Font Loader i Text Geometry import

Sa slike 20, vidljiva su definicije osnovnih elemenata, odnosno scene, kamere i *renderera.*

```
// SCENA, KAMERA I RENDERER
const scene = new THREE.Scene();

const camera = new THREE.PerspectiveCamera(45, window.innerWidth / window.innerHeight, 0.1, 2000);
camera.position.z = 500;
scene.add(camera);

const renderer = new THREE.WebGLRenderer({ antialias: true });
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement);
```

Slika 20: Scena, kamera i renderer

Kao što je vidljivo na slici 21. dodano je ambijentalno svjetlo u svrhu vidljivosti objekata koji će biti kasnije kreirani.

```
// AMBIJENTALNO SVJETLO
const ambientLight = new THREE.AmbientLight(0x666666);
scene.add(ambientLight);
```

Slika 21: Ambijentalno svjetlo

Inicijaliziran je *Texture Loader* radi teksturiranja. U svrhu dobivanja realističnih tekstura svemira i ostalih planetarnih objekata, one će biti preuzete sa stranice čiji je link:

https://www.solarsystemscope.com/textures/

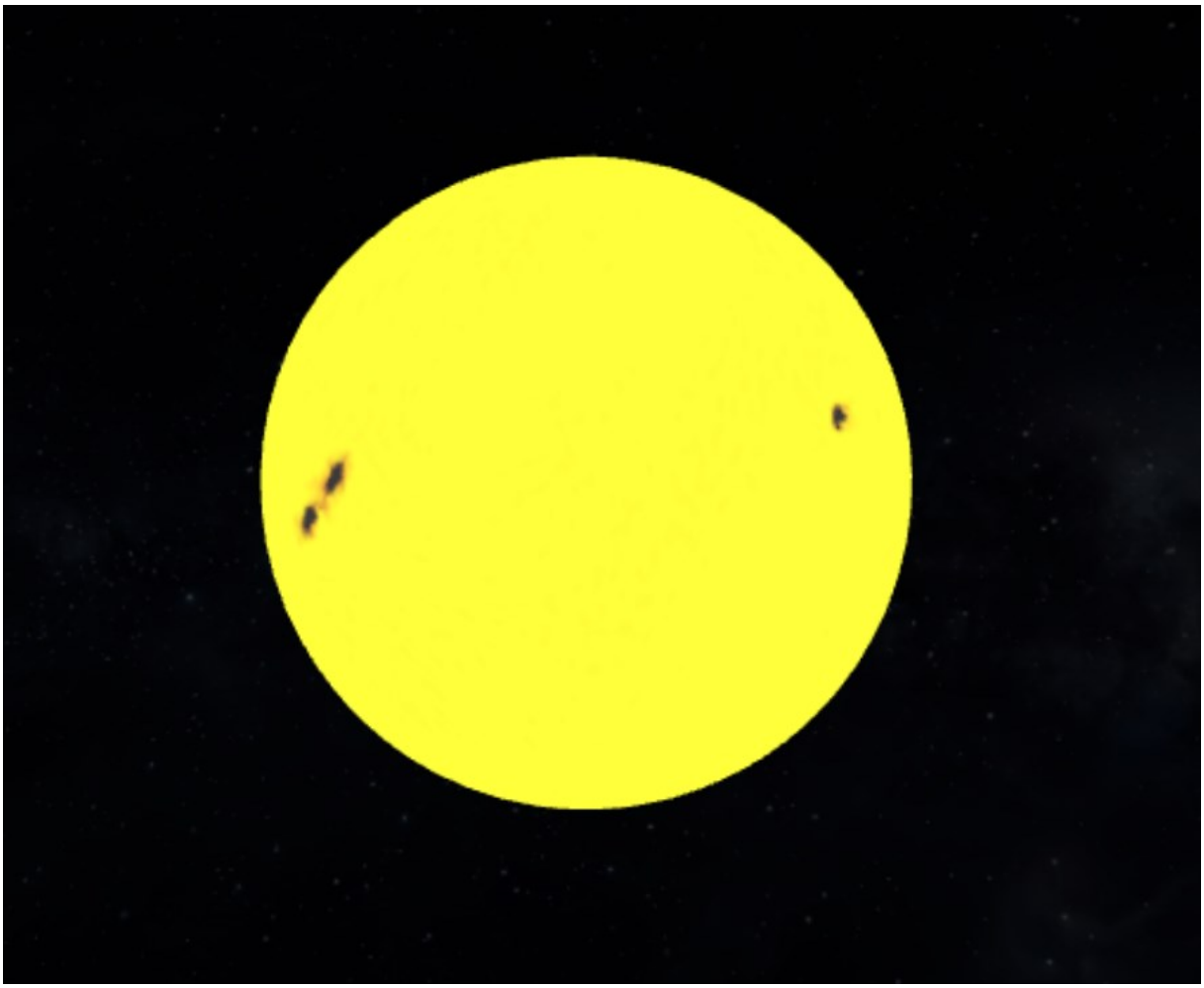
Prva prikazuje prostor svemira i koristi odgovarajući *loader* kroz svojstvo `map` kako bi se dobila tekstura svemira. Svojstvo `side: THREE.DoubleSide` omogućuje da ista tekstura bude i na površini sfere i u njezinoj unutrašnjosti. Druga sfera je Sunce te također koristi odgovarajuću teksturu s ciljem da pridružena tekstura vjerodostojno ilustrira stvarno Sunce. Slika 22. prikazuje programski kod kojim je to izvedeno, a slika 23. dobiveni rezultat teksturiranja.

```
// TEXTURE LOADER
const textureLoader = new THREE.TextureLoader();

// PROSTOR SVEMIRA
const spaceGeometry = new THREE.SphereGeometry(1000, 50, 50);
const spaceMaterial = new THREE.MeshStandardMaterial({
    map: textureLoader.load("svemir.jpg"),
    side: THREE.DoubleSide,
});
const space = new THREE.Mesh(spaceGeometry, spaceMaterial);
scene.add(space);

// SUNCE
const sunGeometry = new THREE.SphereGeometry(50, 32, 32);
const sunMaterial = new THREE.MeshStandardMaterial({
    emissive: 0xffff00,
    emissiveMap: textureLoader.load("sunce.jpg"),
    emissiveIntensity: 1
});
const sun = new THREE.Mesh(sunGeometry, sunMaterial);
sun.position.set(0, 0, 0);
scene.add(sun);
```

Slika 22: Kreiranje prostora svemira i Sunca



Slika 23: Rezultat teksturiranja svemira i Sunca

Slika 24. prikazuje dodavanje točkastog svjetla koje će se umetnuti na istu poziciju gdje se nalazi i Sunce kako bi ono emitiralo sunčevu svjetlost.

```
// TOCKASTO SVJETLO
const pointLight = new THREE.PointLight(0xffffff, 50000, 1000);
pointLight.position.set(sun.position.x, sun.position.y, sun.position.z);
scene.add(pointLight);
```

Slika 24: Točkasto svjetlo

Ostali planetarni objekti stvaraju se po istom principu kao i Sunce, ali se koriste različite pozicije na x koordinati prostora. Sunce se nalazi na nultoj poziciji x, y i z koordinata. Planete je potrebno udaljiti od Sunca te se zbog toga sa svakim planetom povećava vrijednost pozicije planeta na x koordinatu prostora, kao što je vidljivo na slici 25.

```
// MERKUR
const mercuryGeometry = new THREE.SphereGeometry(5, 32, 32);
const mercuryMaterial = new THREE.MeshPhongMaterial({ map: textureLoader.load("merkur.jpg") });
const mercury = new THREE.Mesh(mercuryGeometry, mercuryMaterial);
mercury.position.x = 100;
scene.add(mercury);

// VENERA
const venusGeometry = new THREE.SphereGeometry(8, 32, 32);
const venusMaterial = new THREE.MeshPhongMaterial({ map: textureLoader.load("venera.jpg") });
const venus = new THREE.Mesh(venusGeometry, venusMaterial);
venus.position.x = 150;
scene.add(venus);

// ZEMLJA
const earthGeometry = new THREE.SphereGeometry(8, 32, 32);
const earthMaterial = new THREE.MeshPhongMaterial({ map: textureLoader.load("zemlja.jpg") });
const earth = new THREE.Mesh(earthGeometry, earthMaterial);
earth.position.x = 210;
scene.add(earth);

// MARS
const marsGeometry = new THREE.SphereGeometry(6, 32, 32);
const marsMaterial = new THREE.MeshPhongMaterial({ map: textureLoader.load("mars.jpg") });
const mars = new THREE.Mesh(marsGeometry, marsMaterial);
mars.position.x = 280;
scene.add(mars);

// JUPITER
const jupiterGeometry = new THREE.SphereGeometry(15, 32, 32);
const jupiterMaterial = new THREE.MeshPhongMaterial({ map: textureLoader.load("jupiter.jpg") });
const jupiter = new THREE.Mesh(jupiterGeometry, jupiterMaterial);
jupiter.position.x = 370;
scene.add(jupiter);

// SATURN
const saturnGeometry = new THREE.SphereGeometry(14, 32, 32);
const saturnMaterial = new THREE.MeshPhongMaterial({ map: textureLoader.load("saturn.jpg") });
const saturn = new THREE.Mesh(saturnGeometry, saturnMaterial);
saturn.position.x = 470;
scene.add(saturn);
```

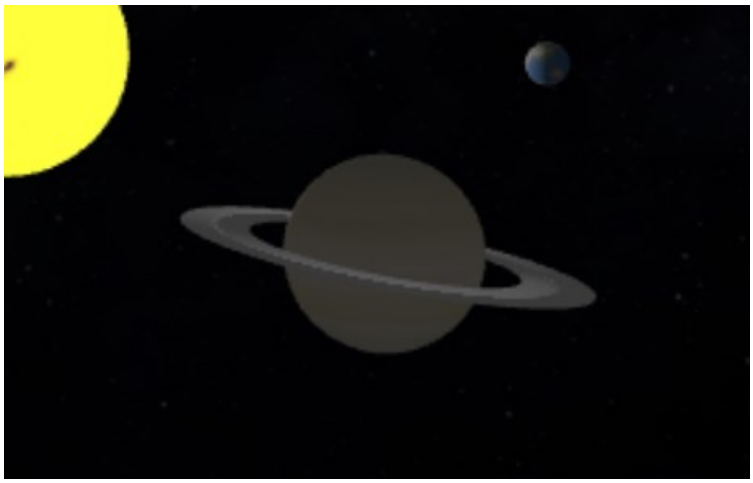
Slika 25: Stvaranje ostalih planetarnih tijela

Slika 26. prikazuje stvaranje Saturnovih prstenova. Postavljaju se na identičnu poziciju na kojoj je Saturn, ali je njegova rotacija na x koordinatu ukošena kako bi se dobio realističniji prikaz planeta:

```
// SATURNOVI PRSTENOVI
const ringGeometry = new THREE.TorusGeometry( 24, 5, 2, 100 );
const ringMaterial = new THREE.MeshPhongMaterial( { colors: 0x336633 } );
const ring = new THREE.Mesh( ringGeometry, ringMaterial );
ring.rotation.x = (Math.PI / 2) * 1.1;
scene.add(ring);
```

Slika 26: Stvaranje Saturnovih prstenova

Slika 27. prikazuje rezultat stvaranje prstena.



Slika 27: Rezultat stvaranja Saturnovih prstenova

Definirana je funkcija koja omogućuje animaciju, konstantno osvježavanje kontrola te kretanje svakog od planeta po njihovim orbitama. Također je za svaki planet (i Sunce) uvedena rotacija oko osi. Nakon ove funkcije dodan je osluškivač koji prilagođava scenu prema omjeru prozora kao što je vidljivo na slici 28.

```
// ANIMACIJA
function animate() {
    requestAnimationFrame(animate);
    // ROTACIJA SVEMIRSKOG PROSTORA
    space.rotation.y += 0.0005;
    // ROTACIJA SUNCA
    sun.rotation.y += 0.001;
    // REVOLUCIJA MERKURA
    mercury.position.x = 100 * Math.sin(Date.now() * 0.0005);
    mercury.position.z = 100 * Math.cos(Date.now() * 0.0005);
    mercury.rotation.y += 0.025;
    // REVOLUCIJA VENERE
    venus.position.x = 150 * Math.sin(Date.now() * 0.00035);
    venus.position.z = 150 * Math.cos(Date.now() * 0.00035);
    venus.rotation.y += 0.0005;
    // REVOLUCIJA ZEMLJE
    earth.position.x = 210 * Math.sin(Date.now() * 0.0002);
    earth.position.z = 210 * Math.cos(Date.now() * 0.0002);
    earth.rotation.y += 0.075;
    // REVOLUCIJA MARSA
    mars.position.x = 280 * Math.sin(Date.now() * 0.00015);
    mars.position.z = 280 * Math.cos(Date.now() * 0.00015);
    mars.rotation.y += 0.05;
    // REVOLUCIJA JUPITERA
    jupiter.position.x = 370 * Math.sin(Date.now() * 0.0001);
    jupiter.position.z = 370 * Math.cos(Date.now() * 0.0001);
    jupiter.rotation.y += 0.1;
    // REVOLUCIJA SATURNA
    saturn.position.x = 470 * Math.sin(Date.now() * 0.00008);
    saturn.position.z = 470 * Math.cos(Date.now() * 0.00008);
    saturn.rotation.y += 0.15;
    // SATURNOV PRSTEN
    ring.position.x = 470 * Math.sin(Date.now() * 0.00008);
    ring.position.z = 470 * Math.cos(Date.now() * 0.00008);

    controls.update();
    renderer.render(scene, camera);
}

animate();

// WINDOW SIZE HANDLER
window.addEventListener('resize', function(){
    let width = window.innerWidth;
    let height = window.innerHeight;
    renderer.setSize(width, height);
    camera.aspect = width / height;
    camera.updateProjectionMatrix();
});
```

Slika 28: Kreiranje ostalih planeta i objekata

Slika 29. prikazuje inicijalizaciju dodatka *Orbit Controls* te je omogućeno glatko usporavanje kretanja kamerom. Uz to je limitirano koliko je moguće približiti se ili udaljiti sceni.

```
// ORBIT CONTROLS
const controls = new OrbitControls(camera, renderer.domElement);
controls.enableDamping = true;
controls.dampingFactor = 0.05;
controls.minDistance = 50;
controls.maxDistance = 950;
```

Slika 29: Orbit Controls

Inicijaliziran je *Font Loader* u svrhu učitavanja preuzetog fonta JSON formata ("Roboto_Regular.json"). Geometrija teksta omogućuje definiciju svojstava teksta, odnosno koji tekst će se ispisati koristeći željeni font, koje veličine će biti, debljine itd.

```
// FONT LOADER
const loader = new FontLoader();

// UČITAVANJE FONTA
loader.load('Roboto_Regular.json', (font) => {
    // KREIRANJE TEKSTUALNOG OBJEKTA
    const textGeometry = new TextGeometry('Solar System', {
        font: font,
        size: 25,
        height: 5,
    });

    const textMaterial = new THREE.MeshBasicMaterial({ color: 0x7755ff });
    const textMesh = new THREE.Mesh(textGeometry, textMaterial);
    textMesh.position.x = -103;
    textMesh.position.y = 100;

    scene.add(textMesh);
});

loader.load('Roboto_Regular.json', (font) => {
    const textGeometry = new TextGeometry('By Nensi', {
        font: font,
        size: 7.5,
        height: 5,
    });

    const textMaterial = new THREE.MeshBasicMaterial({ color: 0x7755ff });
    const textMesh = new THREE.Mesh(textGeometry, textMaterial);
    textMesh.position.x = -25;
    textMesh.position.y = 80;

    scene.add(textMesh);
});
```

Slika 30: Font Loader

Na kraju je dodatno grafičko korisničko sučelje. Cilj je moći promijeniti tri komponente, od kojih su izabrane:

- veličina Sunca
- veličina Saturnovih prstenova
- brzina revolucije Zemlje

Slika 31. prikazuje inicijalizaciju grafičkog korisničkog sučelja i definiciju parametara za promjenu, nakon čega su parametri samo dodani u grafičko korisničko sučelje gdje su definirani svi parametri i funkcije sukladno sa tipom geometrije i animacije.

```
// DAT.GUI
const gui = new dat.GUI();
console.log(gui)

const paramsSun = {
    radius: 50,
};
const paramsRing = {
    radius: 24,
};
const paramsEarth = {
    speed: 0.00020,
};

gui.add(paramsSun, 'radius', 25, 75).name("Size of the Sun").onChange(function (value) {
    updateGeometrySphere();
});
gui.add(paramsRing, 'radius', 20, 50).name("Size of the Ring").onChange(function (value) {
    updateGeometryTorus();
});
gui.add(paramsEarth, 'speed', 0, 0.001).name("Revolucija Zemlje");
```

Slika 31: dat.GUI

Nakon toga izrađene su dvije funkcije. Jedna će omogućiti uništenje geometrije sfere i dodavanje nove sfere koja će primati parametar koji korisnik zada te će se tako mijenjati veličina, odnosno radijus Sunca. Druga funkcija radi isto, ali za radijus Saturnovih prstenova. Nakon funkcija, inicijalizirana je varijabla koja će predstavljati revoluciju Zemlje.

```
function updateGeometrySphere() {
    sun.geometry.dispose();
    sun.geometry = new THREE.SphereGeometry(paramsSun.radius);
}
function updateGeometryTorus() {
    ring.geometry.dispose();
    ring.geometry = new THREE.TorusGeometry(paramsRing.radius , 5, 2, 100 );
}
let revolutionEarth = 0;
```

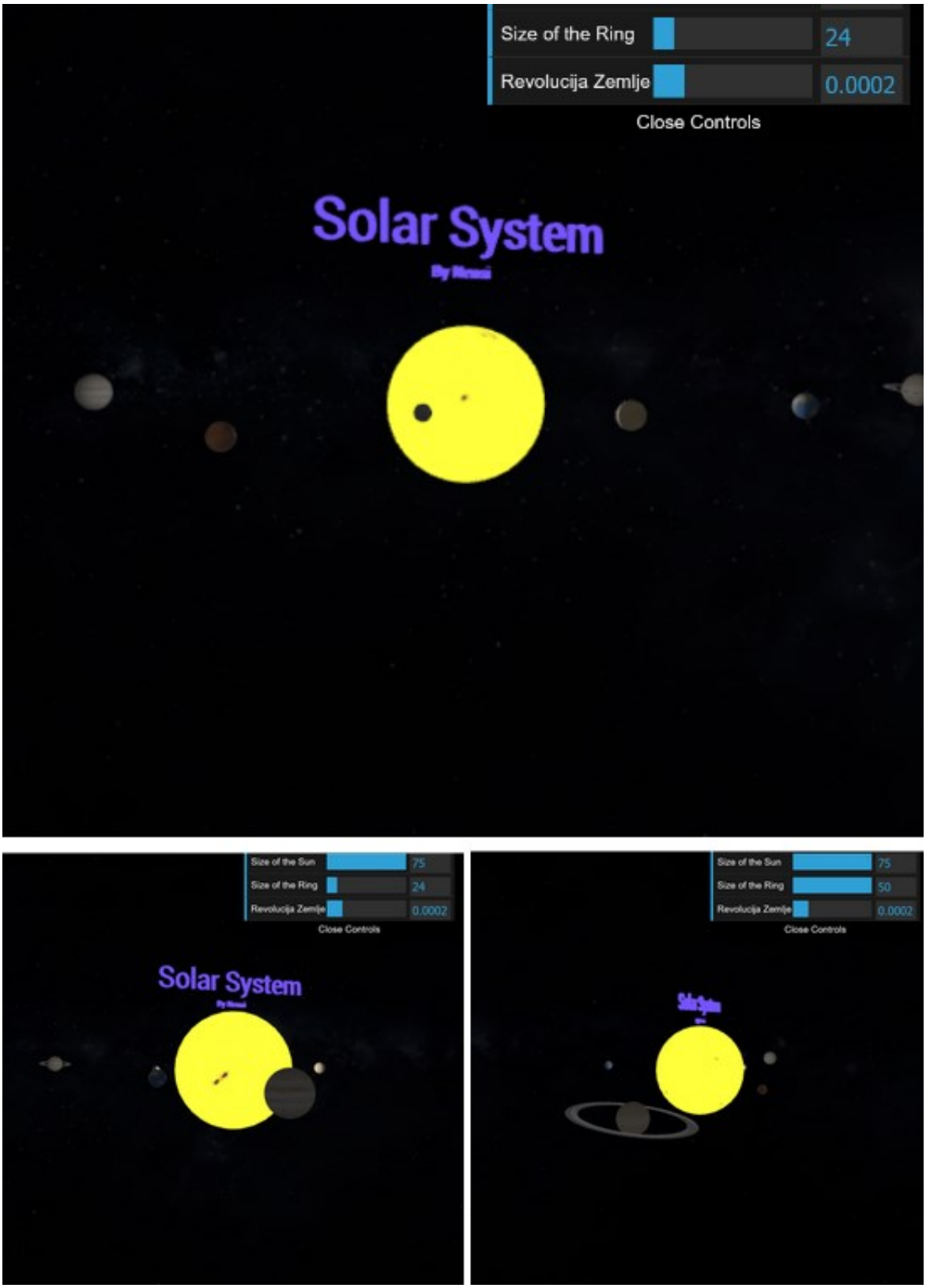
Slika 32: dat.GUI_2

Kako bi se revolucija Zemlje mogla mijenjati, potrebno je modificirati funkciju `animate`. Dodaje se ranije inicijalizirana varijabla te se unutar nje pohranjuje parametar brzine Zemlje koji će se mijenjati ovisno o korisniku. Nakon toga se pod dio funkcije gdje je definirana animacija zemljine orbite umjesto broja brzine stavlja varijabla koja je inicijalizirana pri početku te funkcije. Prikaz funkcije za animaciju prikazana je na slici 33.

```
// ANIMACIJA
function animate() {
    requestAnimationFrame(animate);
    revolutionEarth = paramsEarth.speed;
    // ROTACIJA SVEMIRSKOG PROSTORA
    space.rotation.y += 0.0005;
    // ROTACIJA SUNCA
    sun.rotation.y += 0.001;
    // REVOLUCIJA MERKURA
    mercury.position.x = 100 * Math.sin(Date.now() * 0.0005);
    mercury.position.z = 100 * Math.cos(Date.now() * 0.0005);
    mercury.rotation.y += 0.025;
    // REVOLUCIJA VENERE
    venus.position.x = 150 * Math.sin(Date.now() * 0.00035);
    venus.position.z = 150 * Math.cos(Date.now() * 0.00035);
    venus.rotation.y += 0.0005;
    // REVOLUCIJA ZEMLJE
    earth.position.x = 210 * Math.sin(Date.now() * revolutionEarth);
    earth.position.z = 210 * Math.cos(Date.now() * revolutionEarth);
    earth.rotation.y += 0.075;
    // REVOLUCIJA MARSA
    mars.position.x = 280 * Math.sin(Date.now() * 0.00015);
    mars.position.z = 280 * Math.cos(Date.now() * 0.00015);
    mars.rotation.y += 0.05;
```

Slika 33: dat.GUI_3

Ovime je završena izrada Sunčevog sustava u Three.js-u. Slika 34. prikazuje kako ova aplikacija izgleda na internetskom pregledniku.



Slika 34: Sunčev sustav

ZAKLJUČAK

Three.js je JavaScript biblioteka temeljena na radu WebGL-a te je korištena za svrhe prikaza trodimenzionalne grafike u internetskim preglednicima. Tri najosnovnija elementa Three.js-a su scena, kamera i *renderer*. Pruža niz funkcionalnosti kao što su podešavanja kamere, postavljanje svjetlosti, definicije raznih geometrija, učitavanja modela, fontova, tekstura, a sve s ciljem mogućnosti postizanja što kvalitetnijih i dinamičnijih sadržaja na internetskim preglednicima. Uz to, Three.js nudi i dodatne funkcionalnosti koje je moguće uključiti, a to su *Orbit Controls* i grafičko korisničko sučelje *dat.GUI*. Korisnik pomoću tih dodataka može utjecati izravno na razne elemente scene te ih modificirati bez da mijenjaju sam kod. Three.js uz pravilno kombiniranje svih tih elemenata može omogućiti stvaranje vrlo dinamičnih, bogatih i realističnih trodimenzionalnih scena što se također dokazuje kreiranjem Sunčevog sustava u sklopu završnog rada.

POPIS SLIKA:

POPIS LITERATURE

Educative. Dohvaćeno iz What is dat.GUI in three.js?: https://www.educative.io/answers/what-is-datgui-in-threejs

Three.js - JavaScript 3D Library. Dohvaćeno iz three.js docs: https://threejs.org/docs/

Tutorialspoint. Dohvaćeno iz Three.js Tutorial: https://www.tutorialspoint.com/threejs/index.htm

POPIS IZVORA

[1.] Solar Textures, preuzeto sa Solar System Scope: https://www.solarsystemscope.com/textures/

[2.] 7dir, json-fonts, roboto, preuzeto sa GitHub-a: https://github.com/7dir/json-fonts/tree/master/fonts/cyrillic/roboto