SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4317

3D interakcija u web pregledniku

Josip Mesarić

Zagreb, lipanj 2016.

SADRŽAJ

1. Uvod **1**

1.1. Motivacija 1

1.2. Razvoj modernih web preglednika 1

2. Razrada razvojne okoline i alata **3**

2.1. HTML5: novi vizualni medij 3

2.2. JavaScript programski jezik 3

2.3. Internet preglednik kao platforma 4

2.3.1. JavaScript Virtual Machine(VM) Performance 4

2.3.2. Accelerated Compositing 4

2.3.3. Podrška za animaciju 4

2.4. WebGL - tehnička definicija i detalji 5

2.4.1. WebGL je programsko sučelje (API) 5

2.4.2. WebGL je baziran na OpenGL ES 2.0 5

2.4.3. WebGL u kombinaciji s ostalim sadržajem 5

2.5. JavaScript 3D bibilioteke za WebGL 6

3. Studijski slučaj: Odabir modela za 3D tisak **8**

3.1. Odabir razvojne okoline za WebGL: Three.js 8

3.2. Razvojni proces 9

3.2.1. Blender 9

3.2.2. Three.js 10

3.2.3. HTML5 i CSS3 16

3.2.4. Testiranje web aplikacije pomoću stats.js bibilioteke 16

4. Zaključak **20**

Literatura **21**

1. Uvod

1.1. Motivacija

Živimo u 3-dimenzionalnom svijetu, krećemo se, mislimo i doživljavamo u 3 dimenzije. Većina sadžaja kojem smo izloženi je također u 3 dimenzije(3D), iako su često prikazadni na ravnim 2D ekranima. Animirani filmovi napravljeni na računalu su u 3D-u, online karte nam omogućuju prikaz u 3D okruženju, većina videoigara je u 3D grafici bilo da su pokretani na računalima ili mobilnim telefonima. 3D grafika je stara gotovo kao i samo računalo, čiji korijeni sežu u 1960-te. Korištena je u aplikacijama za inženjerstvo, edukaciju, arhitekturu, financije, prodaju, marketing kao i zabavu. Povijesno gledajući, 3D aplikacije su se uvijek oslanjale na računala visokih performansi aktualnih u to vrijeme, kao i skupu programsku potporu za razvoj istih aplikacija. No, to se promijenilo zadnjih nekoliko godina kada hardwersko procesiranje 3D aplikacija postaje moguće na gotovo svakom računalu ili pametnom telefonu što daje ogroman zamah razvoju 3D grafike. Jednako važno, programi potrebni za pokretanje i renderiranje 3D aplikacija ne samo da su besplatni nego su i univerzalno dostupni, a nazivamo ih *web preglednicima*.

1.2. Razvoj modernih web preglednika

Zadnjih nekoliko godina moderni internetski preglednici su postajali sve moćniji te su se pokazali kao platforme sposobne za pokretanje složenih aplikacija i kompleksne grafike. Većina grafike pokretane kroz internetske preglednike je 2D grafika. Moderni preglednici su prihvatili WebGL standard, koji pruža podršku ne samo za 2D aplikacije i grafiku unutar preglednika, nego omogućuje i izradu prekrasnih 3D aplikacija visokih performansi, koristeći mogućnosti GPU-a.

Programiranje WebGL-a dirketno je vrlo kompleksan posao. Potrebno je poznavanje unutarnjih detalja WebGL-a i učenje kompleksnog *shader* jezika kako bi se is-

koristile sve mogućnosti WebGL-a.Biblioteke poput *three.js* i *babylon.js* omogućuju jednostavan za korištenje JavaScript API, bez potrebe za učenjem svih detalja WebGL-a. Takve bibilioteke nude velik broj funkcionalnosti i API-ja koji se mogu koristiti za stvaranje 3D scena izravno u web pregledniku.

2. Razrada razvojne okoline i alata

2.1. HTML5: novi vizualni medij

HTML je mnogo napredovao od vremena kada je web stranica bila sačinjena od statičnih stranica, formi i pokojeg *buttona*. Ranih 2000-tih preglednici su uveli bogatu interakciju tako što su omogućili web stranici dinamičku izmijenjivost putem *Ajax* tehnika. Unatoč velikom napretku, načini na koje je stranica mogla biti izmijenjena bili su ograničeni grafičkim mogućnostima HTML-a i CSS-a. Ukoliko su programeri htjeli preći te granice, morali su koristiti *plugin-ove* poput *Flasha* i *QuickTimea*.

Posljednih nekoliko godina stvari su se promijenile, nekoliko napredaka unutar web preglednika i pojavio se HTML5. S pojavom HTML5 web preglednici su postali platforma sposobna pokretati sofisticirane aplikacije koje mogu parirati nativnom kodu svojim performansama i funkcionalnostima. HTML5 predstavlja velik iskorak u HTML standardu, uvodeći čišću sintaksu, nove *JavaScript* funkcionalnosti, API-je, mobilnu kompatibilnost i naprednu podršku za multimediju.

2.2. JavaScript programski jezik

JavaScript je lagani, skriptni programski jezik s objektno orijentiranim mogućnostima. Glavna jezgra ovog programskog jezika je ugrađena u sve popularne moderne web preglednike kako bi olakšala programiranje za web s podrškom za objekte koje reprezentiraju prozor web preglednika i njegov sadržaj. Verzija JavaScripta s klijentske strane omogućuje uključivanje izvršnog sadržaja u web stranice - kao posljedica toga, web stranice ne moraju više biti statični HTML, nego mogu uključivati programe koji su u interakciji s korisnikom, kontroliraju preglednik te dinamički stvaraju HTML sadržaj. Sintaksno, jezgra JavaScripta podsjeća na programske jezike C, C++ i Java, s naredbama poput *if* grananja, *while* petlji i *&&* operatora. To je ukratko i jedina sličnost s navedenim programskim jezicima. JavaScript je netipizirani jezik, što znači da varija-

ble ne moraju biti točno određeni tip. Objekti u JavaScriptu su sličniji asocijativnim nizovima u Pearlu nego strukturama u C-u ili objektima u C++-u ili Java-i.Također, objektno orijentirani mehanizam se razlikuje od onoga u C++-u ili Java-i, gdje JavaScript umjesto klasa koristi prototipove.

2.3. Internet preglednik kao platforma

HTML5 donosi napredu grafiku na web; sam po sebi to ne bi uspio bez podrške ostalih važnih napredaka u tehnologiji. Posebice, nekoliko napredaka je utvrdilo put prema razvoju bogatih Internet aplikacija pomoću HTML5:

2.3.1. JavaScript Virtual Machine(VM) Performance

WebGL, kao i Canvas2D, su JavaScript API-ji što znači da će se animacije i interakcije pokretati onoliko glatko i brzo koliko JavaScript u pozadini to dopusti. Prije nekoliko godina virtualne mašine nisu imale dovoljne performanse kako bi omogućile praktičan razvoj 3D aplikacija, za razliku od današnjih JavaScript VM kojima to ne predstavlja nikakav problem.

2.3.2. Accelerated Compositing

Preglednik je zadužen za kombiniranje i kompoziciju različitih elemenata stranice u stvarnom vremenu bez vizualnog raspadanja i ostalih neželjenih efekata. Sadržaj je postao dimnamičniji nego ikada, a preglednici su napravili veliki napredak u savladavnju kompozicije, uključujući i *3D hardware rendering* protokol za sve vizualne elemente, gdje preglednik daje izravne naredbe grafičkoj kartici računala na kojem je animacija pokrenuta.

2.3.3. Podrška za animaciju

Funkcija *requestAnimationFrame()* je uvedena kao alternativa korištenju *setInterval()* ili *setTimeout()* kao pokretačima animacije, što je rezultiralo povećanjem performansi i eliminacijom neželjenih grafičkih pojava (artefakata).

HTML5 preglednici također uključuju funkcionalnosti kao što su višedretveno programiranje, punu podršku za *duplex* TCP/IP mrežno sučelje te lokalno spremanje podataka što omogućuje programerima izradu najmodernijih web aplikacija do sada. Navedene funkcionalnosti, u kombinaciji s WebGL-om, CSS3 3D-om i *Canvasom* pred-

stavljaju revolucionarnu novu platformu za povezane vizualne aplikacije na gotovo svakom računalu ili mobilnom uređaju.

2.4. WebGL - tehnička definicija i detalji

WebGL je višeplatformno sučelje za razvoj aplikacija (API) koje donosi OpenGL ES 2.0 na web kao 3D crtaći kontekst unutar HTML-a, izložen kao sučelje niske razine za *Document Object Model*. Koristi OpenGL *shading* jezik, GLSL ES, te može biti jednostavno kombiniran s ostalim web sadržajem koji se nalazi u slojevima ispod ili iznad 3D sadržaja. WebGL je prikladan za dinamičke 3D web aplikacije u JavaScript programskom jeziku te je podržan u svim vodećim web preglednicima.

2.4.1. WebGL je programsko sučelje (API)

WebGL je dostupan isključivo kroz skup JavaScript-ovih sučelja; ne postoje pripadajući HTML tagovi. 3D renderiranje je analogno 2D crtanju koristeći Canvas element, sve se radi kroz JavaScript API pozive.

2.4.2. WebGL je baziran na OpenGL ES 2.0

OpenGL ES je adaptacija dobro poznatog standarda za 3D renderiranje OpenGL-a. Kratica "ES" označava "embedded system", što znači da je prilagođen malim računalim, uglavnom mobitelima i tabletima. OpenGL ES je API koji pokreće 3D grafiku na iOS i Android operacijskim sustavima. Tvorci WebGL-a su osjećali da, bazirajući API na "laganom" OpenGL ES standardu, mogu dobiti bolje performanse i iskoristivost kod web aplikacija koje se pokreću na više platformi i u više različitih preglednika.

2.4.3. WebGL u kombinaciji s ostalim sadržajem

WebGL slojevi se nalaze povrh ili ispod ostalog sadržaja na stranici. 3D canvas u kojem se prikazuje grafika može zauzeti samo dio stranice kao i cijelu stranicu. Može se nalaziti unutar HTML-ovih tagova kojima se dodijeljuje Z-vrijednost. Znači da možemo razviti vlasitu grafiku koristeći WebGL, ali svi ostali elementi stranice su i dalje stvoreni koristeći HTML. Preglednik naposlijetku radi kompoziciju elemenata na stranici prema zadanim HTML okvirima.

2.5. JavaScript 3D bibilioteke za WebGL

Zahvaljujući naprednim JavaScript API-jima poput WebGL-a, moderni web preglednici su u mogućnosti prikazivati napredne 2D i 3D grafike bez potrebe za dodatnim *plug-inovima*. Pamtno raspoređujući resurse dodijeljene grafičke kartice, WebGL daje web stranicama mogućnost pristupa dinamičkom *shadingu* i realističnim fizikama.

Oduvijek popularni *Three.js* kao i noviji *Babylon.js* nude web programerima apstraktne temelje za kreiranje kompleksnih WebGL aplikacija.

Početak *Three.js* seže još u travanj 2009. kada je originalno napisan u *ActionScript* jeziku prije nego je preveden u *JavaScript* jezik. Kako je napisan prije pojave WebGL-a, *Three.js* ima specifičnu mogućnost prebacivanja sučelja za prikaz između *SVG* i *HTML5 canvas* elementa kao dodatka WebGL-u.

Babylon.js je nešto mlađi, pojavio se sredinom 2013. Stvoren od strane Microsofta, *Babylon.js* je predstavljem zajedno s podrškom *Internet Explorera 11* za WebGL API. I jedna i druga razvojna okolina su pod licencom *open source*.

Three.js i *Babylon.js* razvojne okoline nude biblioteke koje su jednostavne za korištenje za baratanje funkcionalnostima WebGL animacija. Obje bibilioteke koriste slične metode za upravljanje WebGL-om, koristeći scene, kamere, *rendere* i modele za animiranje. Upotreba navedenih mogućnosti unutar HTML-a svodi se na jednu liniju koda u kojoj se navodi povezivanje s odgovarajućim *JavaScript* datotekom. Specifičnost *Babylon.js* radne okoline je u tome da zahtijeva uključivanje i *open source* bibilioteke *Hand.js*.

Three.js

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

Babylon.js

```
<script src="babylon.js"></script>
<script src="hand.js"></script>
```

Glavna razlika između ove dvije razvojne okoline je upravo u različitosti problema za koje su namijenjene. Naravno, time se ne isključuje mogućnost da se ista 3D animacija dobije korištenjem oba rješenja, ali svaka razvojna okolina je napravljena kako bi olakšala stvaranje određenog tipa grafičkih web aplikacija.

Three.js je stvoren s ciljem što lakšeg iskorištavanja web baziranih renderera za stvaranje 3D grafika i animacija pokretanih grafičkom karticom(GPU). Kao posljedica, dobivena je razvojna okolina koja pokriva velik broj različitih pristupa grafici unutar web preglednika bez fokusiranja na jednu animacijsku tehniku. Fleksibilnost

dizajna čini *Three.js* odličnim alatom za web grafiku opće namjene poput logotipova, aplikacija za modeliranje, prikaza grafičkih modela.

Dok *Three.js* nudi širok spektar mogućnosti i pristupa razvoju grafike na webu, nije se specijalizirao kako bi maksimalno olakšao razvoj određenog tipa grafičkih aplikacija. Originalno napravljen kao *game engine* unutar *Silverlight* razvojne okoline, *Babylon.js* ima ciljan pristup. Specijaliziran je za razvoj videoigara pokretanih WebGL-om unutar internet preglednika. Nudi mogućnosti poput detekcije sudara (*collision detection*) i *antialiasinga*, funkcionalnosti koje su od izuzetne važnosti za *game development*.

3. Studijski slučaj: Odabir modela za 3D tisak

3D tisak je postao zadnjih godina rastući trend u svijetu. Zahvaljujući sve nižoj cijeni samih uređaja, dostupnosti materijala te jednostavnosti izrade 3D modela od raznih materijala postali su široko dostupni te sve prisutniji u kućnoj upotrebi. Do sada su bili korišteni u proizvodnim procesima raznih industrija gdje su se koristili za proizvodnju složenih dijelova (autoindustrija, strojarstvo, dizajn, elektronika, automatika) prethodno napravljenih u nekom od alata za 3D modeliranje.

Zadatak je napraviti web aplikaciju za tvrtku koja se bavi 3D tiskom. Na ulaznoj strani trebaju biti ponuđeni neki proizvodi koje je moguće otisnuti. Odabirom određenog proizvoda dolazi se na stranicu s detaljnijim modelom proizvoda kao i dodatnim informacijama. Za prikaz 3D modela proizvoda koristit će se WebGL.

3.1. Odabir razvojne okoline za WebGL: Three.js

Odabrana implementacija za WebGL je Three.js. Usporedbom Three.js-a i Babylon.js-a došao sam do zaključka da je Three.js elegantnije rješenje za dobiveni zadatak. Naime, Babylon.js je radni okvir namijenjen za razvoj web igara s podrškom za napredne funkcije važne za podizanje performansi videoigara (antialiasing, collision detection). Three.js je namijenjen široj upotrebi te je zbog toga jednostavniji za korištenje kada nisu potrebne kompleksne scene i fizika unutar njih. Three.js se pokazao kao dobra opcija zbog svoje jednostavnosti i kompaktnosti, pogotovo u slučajevima kada ga je potrebno ukomponirati unutar drugih HTML elemenata web stranice. Three.js je dostupan u dvije verzije:

1. **Three.min.js:** Ovu biblioteku ćemo normalno koristiti kada postavljamo Three.js stranice na internet. Ovo je minimizirana verzija Three.js-a, stvorena koristeći *UglifyJS*, koja je pola veličine klasične Three.js bibilioteke.

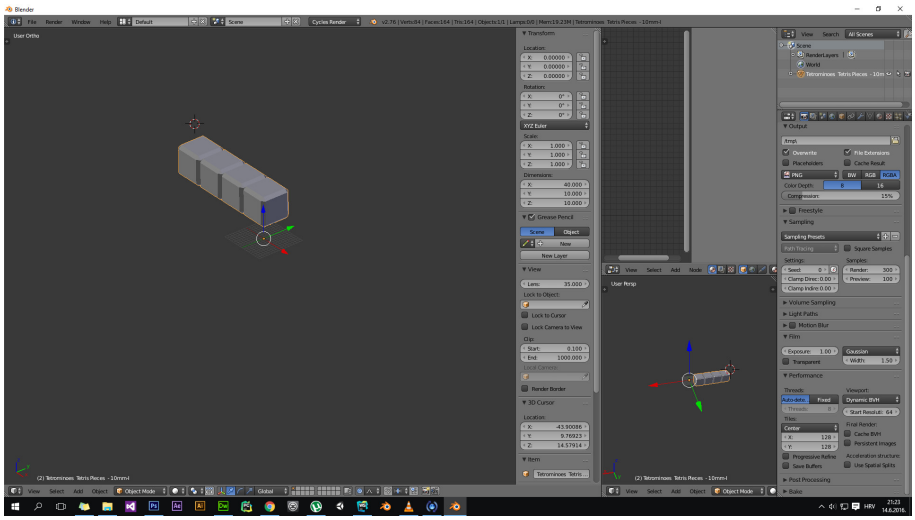
2. **Three.js:** Ovo je klasična Three.js biblioteka. To je biblioteka koju je preporučeno koristiti u razvojnom procesu, jer olakšava *debuggiranje* tako što je Three.js izvorni kod puno čitljiviji i pregledniji.

3.2. Razvojni proces

3.2.1. Blender

Blender je *open-source* software koji nudi skup alata za stvaranje kompletnih 3D-grafika. Uključuje alate za modeliranje, teksture, *shading*, animacije, compozicije, renderiranje, video editiranje i još mnogo toga. Od razvoja verzije 2.50, u kojoj je uvedeno potpuno novo korisničko sučelje (UI), Blenderova baza korisnika je značajno porasla. Blender su počeli koristiti studiji za animaciju za produkciju filmova najviše kvalitete.

Ciljano tržište su profesionalci, slobodni 3D umjetnici te mali studiji. Još uvijek nije široko prihvaćen od strane velikih studija zbog više razloga. Veliki studiji uglavnom koriste već dugo vremena isti software koji ima impresivne *third-party plug in-ove* koji su razvijeni tijekom godina za specifičnu upotrebu unutar produkcijskog procesa. Blender je alat koji se još razvija i unatoč tome što je još relativno novi software na sceni, nadilazi te probleme te ga veći studiji malo po malo uvode u svoje radne procese kao što su modeliranje i UV odmotavanje (*unwrapping*, dva područja u kojima je Blender izuzetno efikasan.



Slika 3.1: Program za 3D modeliranje - Blender

3.2.2. Three.js

Uvoz gotovih 3D modela

Three.js nudi podršku za velik broj 3D formata(JSON, OBJ i MTL, Collada, STL, CTM, VTK, PDB, PLY). Za svaki od tih formata je potrebno uključiti dodatnu JavaScript datoteku koje se nalaze unutar *examples/js/loaders* direktorija Three.js distribucije.

OBJ i MTL format

OBJ i MTL su komplementarni formati koji se često koriste zajedno. OBJ datoteka definira format i geometriju dok MTL datoteka definira materijale koji su korišteni. Oba formata OBJ i MTL su formati bazirani na tekstu. Dio OBJ formata se vidi u sljedećem dijelu koda:

```
v -0.32442 0.010796 0.025935
v -0.028519 0.013697 0.026201
v -0.029086 0.014533 0.021409
usemtl Material
s 1
f 2731 2735 2736 2732
f 2732 2736 3043 3044
```

Dok je MTL datoteka definira materijale na sljedeći način:

```
newmtl Material
Ns 56.862745
Ka 0.000000 0.000000 0.000000
Kd 0.360725 0.227524 0.127497
Ks 0.010000 0.010000 0.010000
Ni 1.000000
d  1.000000
illum 2
```

OBJ i MTL su formati koje Three.js dobro razumije, a podržava ih i Blender. Modeli će iz Blendera biti izvezeni u OBJ i MTL formatu te u Three.js učitani pomoću pripadajućih *loadera*. Three.js nudi dva različita loadera na raspolaganje. Ukoliko želimo uvesti samo geometriju možemo koristiti OBJLoader. Kako bismo učitali OBJ model u Three.js moramo dodati *ObjLoader.js* datoteku:

```
<script type="text/javascript"
    src="../libs/OBJLoader.js"></script>
```

Uvoz modela iz OBJ formata:

```
var loader = new THREE.OBJLoader();
loader.load('../models/model.obj', function (loadedMesh){
 var material = new THREE.MeshLambertMaterial(
 {color: 0x5C3A21});
 loadedMesh.children.forEach(function (child) {
   child.material = material;
    child.geometry.computeFaceNormals();
        child.geometry.computeVertexNormals();
    });
mesh = loadedMesh;
loadedMesh.scale.set(100, 100, 100);
loadedMesh.rotation.x = -0.3;
scene.add(loadedMesh);
    });
```

U ovom kodu koristimo *OBJLoader* klasu za učitavanje modela iz URL-a. Jednom kada je model učitan, povratni poziv je pokrenut i dodajemo model u scenu. Čest je slučaj s klasama za učitavanje geometrija da prikazuju geometrije kao hijerarhiju grupa i geometrija. Potrebno je razumijevanje te hijerarhije kako bi se što jednostavnije smjestili i primjenili ispravni materijali.

Za učitavanje OBJ i MTL formata koristit ćemo *OBJMTLLoader* klasu. Na taj način ćemo učitati model i izravno primijeniti pripadajući materijal. Prvo dodajemo potrebne datoteke:

```
<script type="text/javascript" src="../libs/
OBJLoader.js"></script>
<script type="text/javascript" src="../libs/
MTLLoader.js"></script>
<script type="text/javascript" src="../libs/
OBJMTLLoader.js"></script>
```

Modele iz OBJ i MTL formata učitavamo na sljedeći način:

```
var loader = new THREE.OBJMTLLoader();
loader.addEventListener('load', function (event) {
```

```
var object=event.content;
object.scale.set(140, 140, 140);
    object.rotation.x = 0.2;
    object.rotation.y = -1.3;
    scene.add(object);
    });
loader.load('../models/model.obj','../models/model.mtl');
```

Za ovaj *loader* je potrebno deklarirati *event listener* za *load* događaj. Kada model, materijal i teksture budu učitani, ovaj *listener* je pozvan. Ovdje određujemo i rotaciju objekta te naposlijetku dodajemo model u scenu.

Grupiranje objekata

Kada je mesh stvoren iz geometrije koja koristi nekoliko materijala(ili kada je učitan iz nekog od gotovih formata), Three.js stvara grupu. U tu grupu su dodane višestruke kopije geometrije, svaka sa svojim specifičnim materijalom. Povratna grupa izgleda kao mesh koji koristi različite materijale, dok je to u stvarnosti grupa koja sadrži nekoliko mesh-eva. Stvaranje grupa u Three.js-u je srećom vrlo jednostavno. Svaki mesh koji stvorimo može sadržavati druge elemente kao svoju djecu, a dodajemo ih koristeći naredbu *add*. Rezultat takve strukture je jednostavnije baratanje grupom mesh-eva gdje pomicanje, skaliranje ili rotiranje roditeljskog objekta rezultira istom akcijom primjenjenom na svu djecu tog objekta. Naravno, iako su grupirani, svaki od objekata unutar grupe je moguće zasebno referencirati i modificirati kao individualnu geometriju.

Odabir modela mišem

Važan dio aplikacije je odabir modela mišem unutar HTML5 canvasa na čiji ulaz je preusmjeren izlaz WebGL-a u kojem se prikazuje 3D model. U nastavku je prikazan odsječak koda zadužen za odabir objekta.

```
function onDocumentMouseDown(event) {
 event.preventDefault();
 var vector = new THREE.Vector3(( event.clientX /
 window.innerWidth ) * 2 - 1,
 -( event.clientY / window.innerHeight ) * 2 + 1, 0.5);
 vector = vector.unproject(camera);
 var raycaster = new THREE.Raycaster(camera.position,
```

```
vector.sub(camera.position).normalize());
var intersects = raycaster.intersectObjects
                 ([mesh,mesh2,mesh3], true);
if (intersects.length > 0) {
   console.log(intersects[0]);
   intersects[0].object.material.transparent = true;
   intersects[0].object.material.opacity = 0.1;
       } }
```

U ovom kodu koristimo THREE.Projector klasu zajedno s THREE.Raycaster klasom kako bismo odredili jesmo li kliknuli na određeni 3D model. Redoslijed operacija nakon što kliknemo negdje na ekranu:

1. Prvo, je stvoren vektor temeljem pozicije na ekranu na koju smo kliknuli
2. Iduće, pomoću funkcije *vector.unproject* obavimo konverziju odabrane pozicije na ekranu u koordinate naše Three.js scene
3. Koristimo *THREE.Raycaster* objekt (koji je vratila *projector.pickingRay* funkcija) kako bi poslali zraku u scenu iz mjesta na koje smo kliknuli na ekranu
4. Konačno, koristimo funkciju *raycaster.intersectObjects* kako bismo odredili je li zraka presjekla neki od objekata u sceni na svojem putu

Rezultati posljednjeg koraka sadržavaju informacije o svakom objektu kojeg je dodirnula zamišljenja zraka. Sljedeće informacije su dostupne:

```
distance: 49.123050
face: THREE.Face3
object: THREE.Mesh
point: THREE.Vector3
```

Svojstvo *object* je mesh na koji smo kliknuli, *face* i *faceIndex* pokazuju na plohu mesha kojeg smo odabrali. Svojstvo *distance* mjeri udaljenost od kamere do odabranog objekta dok je *point* točna lokacija na mesh-u koju smo kliknuli.

Animiranje pomoću requestAnimationFrame() metode

Ukoliko želimo animirati scenu prva stvar koju trebamo učiniti je naći način kako renderirati scenu u specifičnom vremenskom intervalu. Prije pojave HTML5 i pripadajućih JavaScript API-ja, taj problem se riješavao pomoću *setInterval(function,interval)*

funkcije. Pomoću *setInterval()* metode možemo odrediti da funkcija bude pozvana svaki puta kada prođe zadani vremenski interval. Problem kod te funkcije je to što ne uzima u obzir što se događa u internet pregledniku u kojem je pokrenuta. Pa tako ona poziva funkciju čak i ukoliko je otvoren drugi tab. Osim toga *setInterval()* metoda nije sinkronizirana s iscrtavanjem ekrana što rezultira povećanim opretećenjem CPU-a i lošijim performansama.

Moderni preglednici srećom imaju rješenje za navedene probleme s funkcijom *setInterval()*: funkcija *requestAnimationFrame()*. Pomoću *requestAnimationFrame()* funkcije možemo zadati da funkcija bude pozvana u intervalima koje određuje sam preglednik. Sve promjene i pokretače animacije možemo smjestiti u jednu funkciju, za koju će se sam preglednik pobrinuti da bude prikazana na najefikasniji način.

Sve što je potrebno učiniti jest kreirati funkciju koja je zadužena za renderiranje, kao što je prikazano u nastavku.

```
function renderScene(){
renderer.render(scene, camera);
requestAnimationFrame(renderScene);
}
```

Unutar *renderScene()* funkcije zovemo requestAnimationFrame() metodu i na taj način održavamo tijek animacije. Možemo primjetiti da u ovom odsječku koda funkciju *render()* moramo pozvati samo jednom, i to kada smo završili s incijalizacijom scene. Unutar same *render()* funkcije koristimo *requestAnimationFrame* kako bismo zakazali iduće renderiranje. Na taj način će se web preglednik pobrinuti da *render()* funkcija bude pozvana u ispravnom vremenskom intervalu(uglavnom je to oko 60 puta u sekundi). Već je spomenuto da prije dodavanja *requestAnimationFrame* funkcije u moderne preglednike, koristile su se funkcije *setInterval(function,interval)* ili *setTimeout(function, interval)*. One su osiguravale pozivanje zadane funkcije nakon što istekne zadani vremenski interval. Problem s tim pristupom je u tome što ne uzima u obzir ostale događaje i okolinu u kojoj se izvršava, pa se izvršava renderiranje čak i ukoliko je korisnik u drugom tabu te na taj način nepotrebno troši resurse. Drugi problem s tim pristupom je taj što se se funkcija poziva kada je to zadano, a ne kada je najbolji trenutak za preglednik da pozove funkciju ponovnog iscrtavanja. Još jednom, to rezultira povećanim opterećenjem procesora. Koristeći funkciju *requestAnimationFrame* ne govorimo pregledniku kada da osvježi prikaz, nego tražimo od preglednika da pokrene zadanu funkciju kada je za to najbolji trenutak. Uglavnom to rezultira *frame rate-om* od oko 60fps-a. Nakon što smo kreirali scenu i dodali sve elemente, pozivamo

renderScene() kako bismo pokrenuli animaciju:

```
...
$("#WebGL-output").append(renderer.domElement);
renderScene();
```

Koristeći gore navedeni pristup vrlo je jednostavno animirati objekte mijenjajući njihovu rotaciju, veličinu, materijale, vrhove i sve ostale atribute koje može objekt posjedovati. Sljedeći primjer *render* funkcije(petlje) pokazuje animaciju upravo promjenom jednog od atributa objekta. Sve što je potrebno učiniti je promijeniti jedan od atributa zadanog *mesh-a*, dok Three.js odrađuje ostatak posla za nas:

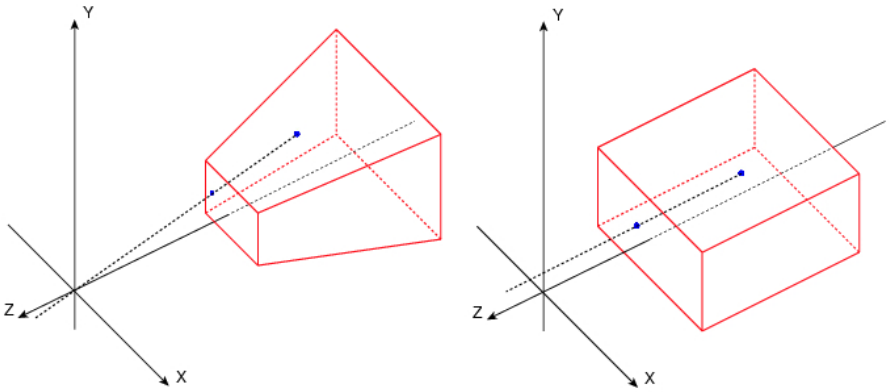
```
function render() {
 stats.update();
 if (mesh) {
  mesh.rotation.y += 0.006;
  mesh.rotation.x += 0.006;
  mesh2.rotation.y += 0.006;
  mesh2.rotation.x += 0.006;
  mesh3.rotation.y += 0.006;
  mesh3.rotation.x += 0.006;
  }
 webGLRenderer.render(scene, camera);
 requestAnimationFrame(render);
 }
```

U navedenom primjeru možemo vidjeti kako dobiti animaciju tijela mijenjajući njenu rotaciju po x i y osi s malim pomakom.

Odabir kamere:ortografska vs. perspektivna kamera

Dva različita tipa kamera su podržane unutar Three.js biblioteke: ortografska i perspektivna. Razlika između ortografskog i perspektivnog prikaza je prikazana na sljedećoj slici.

Ortografski prikaz ima fiksiranu dubinu i svi likovi su jednakih dimenzija, neovisno o udaljenosti od kamere. Kutevi i paralelnosti su očuvani. U perspektivnom prikazu možemo vidjeti udaljenosti od kamere, scena ima dubinu. Mi ćemo koristiti perspektivnu kameru, jer najbolje prikazuje stvarni svijet.



Slika 3.2: Perspektivna i ortografska kamera

```
camera = new THREE.PerspectiveCamera
(45,window.innerWidth/ window.innerHeight, 0.1, 1000);
```

Argumenti THREE.PerspectiveCamera funkcije uz objašnjenja:

Ograničenje Three.js-a: Nemogućnost dijeljenja resursa između više izlaza

Kako bi što lakše postavili strukturu web stranice, prigodno je koristiti nekoliko različitih WebGL canvasa na koje se preusmjeruje izlaz WebGL renderera. Nažalost, Three.js nema mogućnost dijeljenja objekata i resursa unutar različitih renderera. Upravo zbog tog ograničenja potrebno je raditi posebne scene, loadere, učitavati objekte te ih dodavati zasebnom rendereru koji se dodijeljuje točno jednom HTML elementu.

3.2.3. HTML5 i CSS3

Osnovna struktura stranice je postavljena u HTML5 korištenjem WebGL canvasa i standardnih HTML elemenata. Pozicioniranje i dizajn je odrađen u CSS-u.

3.2.4. Testiranje web aplikacije pomoću stats.js bibilioteke

Kao pomoć pri testiranju koristit ćemo malu pomoćnu bibilioteku koja nam nudi informacije o broju prikazanih slika u sekundi kojim se zadana animacija vrti. Ova biblioteka, napravljena od istoga autora kao i sam Three.js, renderira mali graf koji daje informacije o FPS-u (Frames Per Second) koji dobivamo za animaciju koja se pokreće. Kako bismo dodali statistiku, prvo moramo uključiti biblioteku u HTML <header> tagu:

Tablica 3.1: Argumenti

Argument	Opis
fov	field of view. Ovo je dio scene koji kamera vidi iz svoje pozicije. Na primjer, ljudi imaju gledište od skoro 180 stupnjeva dok neke ptice imaju gledište od skoro 360 stupnjeva. Kako klasični zasloni ne nude potpuni pregled ljudskog oka, uglavnom koristimo manje vrijednosti. Za igre se koristi kut od 60 do 90 stupnjeva dok se 45 stupnjeva smatra dobro zadanom vrijednošću.
aspect	Predstavlja odnos između vertikalne i horizontalne veličine prozora u kojem se prikazuje izlaz. Aspect omjer određuje razliku u prikazu horizontalnog polja prikaza i vertikalnog polja prikaza. U ovoj aplikaciji ćemo koristiti cijeli dodijeljeni prozor za prikaz, pa upravo njegove dimenzije stavljamo kao omjer.
near	Argument koji definira koliko blizu kamere će Three.js biblioteka renderirati scenu. Uglavnom ovdje postavljamo vrlo malu vrijednost kako bi renderirali sve što kamera može prikazati sa svoje pozicije.
far	Argument koji definira koliko daleko kamera vidi sa svoje pozicije. Ukoliko ovu vrijednost namjestimo na premalu vrijednost, neki dijelovi scene mogu ne biti prikazani. Ukoliko je vrijednost prevelika to može naštetiti performansama zbog prevelikog obujma koji kamera prikazuje.

Slika 3.3: HTML struktura web stranice

```
<script type="text/javascript"
src="../libs/stats.js"></script>
```

Zatim dodajemo element koji će se koristiti kao izlaz za prikazivanje grafa sa statistikom:

```
<div id="Stats-output"></div>
```

Posljednja stvar koju moramo učiniti je inicijalizirati statistiku i dodati ju navedenom elementu:

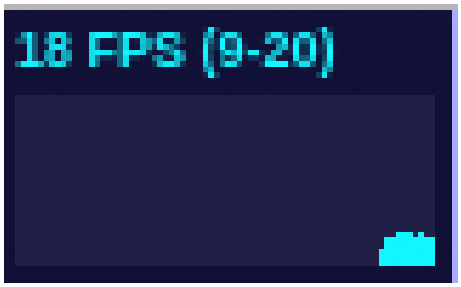
```
function initStats(){
 var stats = new Stats();
 stats.setMode(0);
 stats.domElement.style.position = 'absolute';
 stats.domElement.style.left = '0px';
 stats.domElement.style.top = '0px';
 $("#Stats-output").append( stats.domElement );
 return stats;
}
```

Ova funkcija inicijalizira statistiku. Zanimljiv dio je *setMode()* funkcija. Ukoliko stavimo vrijednost 0 mjerit ćemo broj slika u seknudi (FPS), a ukoliko stavimo vrijednost na 1 mjerit ćemo vrijeme renderiranja pojedinog prikaza. U ovom primjeru, interesantniji nam je podatak broj slika u sekundi (FPS), pa prenosimo funkciji setMode() vrijednost 0. Na početku naše anonimne jQuery funkcije, pozivamo funkciju za statistiku.

```
$(function () {
 var stats = initStats();
...
}
```

Posljednja stvar koju moramo učiniti je dojaviti *stats* objektu kada smo u novom ciklusu renderiranja. To možemo učiniti pozivom metode *stats.update()* unutar *render()* funkcije:

```
function render() {
 stats.update();
 ...
 requestAnimationFrame(render);
 renderer.render(scene, camera);
 }
```



Slika 3.4: stats.js

4. Zaključak

Današnja računala, a i prijenosni uređaji, postali su dovoljno moćni da mogu bez problema pokretati naprednu grafiku unutar web preglednika. Kao vrlo važan standard u tom području se pokazano WebGL koji donosi sve mogućnosti i prednosti OpenGL-a unutar web preglednika.

Obzirom da je WebGL izuzetno kompleksan za razvijanje korisničkih aplikacija napravljene su pomoćne biblioteke koje olakšavaju rad s WebGL-om i skrivaju nepotrebne detalje od programera. Jedna od takvih bibilioteka je Three.js, koja olakšava programiranje WebGL standarda u JavaScript programskom jeziku. Three.js bitno olakšava posao programiranja grafičkih web aplikacija, ali to ne znači da je savršen i razvijen do kraja. Jedan od uočenih nedostataka Three.js-a je nemogućnost dijeljenja resursa između više različitih grafičkih izlaza unutar jedne web stranice. Time se povećava memorijska i procesorska zahtjevnost aplikacije.

LITERATURA

Jos Dirksen. *Learning Three.js: The JavaScript 3D Library for WebGL.* Packt Publishing Ltd., Birmingham B3 2PB,UK, 2013.

Tony Parisi. *Programming 3D Applications with HTML5 and WebGL.* O'Reilly Media, 2014.

3D interakcija u web pregledniku

Sažetak

WebGL je bibilioteka za hardwerski pokretano 3D renderiranje pomoću JavaScripta. Bazirano je na dokazanom grafičkom API-ju OpenGL. WebGL je podržan od gotovo svih web preglednika kao i većeg dijela mobilnih preglednika. Three.js je biblioteka koja olakšava razvoj aplikacija u WebGL standardu i programskom jeziku JavaScript.

Ključne riječi: WebGL, JavaScript, 3D tisak, Three.js, web aplikacije

3D interaction in a web browser

Abstract

WebGL is library for hardware-rendered 3D graphics using JavaScript language. It is based on well known graphic API Open GL. WebGL is supported by all modern browsers and mobile devices. Three.js is WebGL library for JavaScript, which make developing in WebGL much easier.

Keywords: WebGL, JavaScript, 3D print, Three.js, web applications