SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

DIPLOMSKI RAD br. 561

# Grafički objekti u Web preglednicima

Ana Nekić

Zagreb, lipanj 2013.

***Zahvala***

*Zahvaljujem se  mentorici prof. dr. sc. Željki Mihajlović na susretljivosti, pomoći i savjetima pri izradi ovog rada.*

## Sadržaj

# 1. Uvod

HTML5 kratica je za najnoviju inačicu *Hypertext Markup Language,* tj. jezika za označavanje sadržaja koji se prikazuju u Web preglednicima. Standard HTML5 objavljen je od strane W3C-a (*eng. World Wide Web Consortium*) 2008. godine, ali u širu upotrebu ulazi tek 2011. godine. Danas je podržan od strane najkorištenijih Web preglednika kao što su Firefox, Internet Explorer, Chrome, Opera, itd.

HTML5 je nasljednik prethodnog standarda HTML4, objavljenog 1999. godine. S obzirom na veliki opseg promjena koje su se dogodile u percepciji i korištenju Interneta, HTML5 standard je zadržao odrednice prethodnog HTML4 standarda i prihvatio zahtjeve suvremene tehnologije. Značajke standarda HTML5 zasnivaju se na HTML-u, CSS-u, DOM sučelju i programskom jeziku JavaScript. Standard HTML5 trebao bi biti neovisan o uređajima na kojima se koristi, te kao takav pogodan za izvođenje na računalima, pametnim telefonima i tabletima. Zastupa se javna dostupnost, tj. vidljivost implementacijskog procesa.

Standard HTML5 uvelike olakšava proces implementacije, ponajviše zbog izražene intuitivnosti. Najviše novosti u odnosu na prethodne standarde prisutno je u području multimedije. U navedenu svrhu u standard HTML5 dodani su elementi <canvas>, <audio> i <video>. Za razvoj aplikacija iz područja računalne grafike najznačajniji je element – platno (*eng. canvas*)[3]. Naime, platno podržava tzv. vektorsku grafiku. Vektorsku grafiku karakterizira korištenje grafičkih primitiva kao što su točke, linije, krivulje i površine, koje se definiraju pripadnim matematičkim izrazima. Stoga se grafičke aplikacije u preglednicima izvode brže i efikasnije, bez potrebe za dodatnim implementacijskim aplikacijama.

U sklopu HTML5 standarda moguće je definirati prostor za komentare, članke ili postove, koji je neovisan o sadržaju stranice pomoću oznake <article>[1]. Zaglavlje (*eng. header*) i podnožje (*eng. footer*) stranice više nije potrebno specijalno identificirati, već se za njih koriste nove oznake <header>, odnosno <footer>[2]. Prilikom odjeljivanja poglavlja moguće je koristiti za to namijenjenu novu oznaku <section>. Pomoću oznake <aside> odjeljuje se sadržaj dokumenta od dodatno umetnutih dijelova i koristi se za implementaciju rubnog stupca (*eng. side bar*).

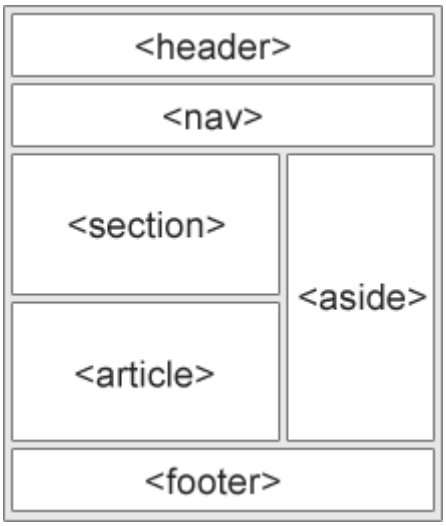
# 2. HTML5

HTML5 karakteriziraju brojni novi elementi, kojima je moguće proširivati sadržaj koji se prikazuje u Web pregledniku. HTML5 podržava MP4, Ogg i WebM video formate[4]. Od audio formata moguće je koristiti MP3, Wav i Ogg[5]. Pripadni <video> i <audio> elementi koriste se za dodavanje video i audio sadržaja, a moguće ih je i tekstualno opisati pomoću oznake <track>. Osim spremnika za pohranu dodatnog sadržaja <embeded>, u medijske se elemente ubraja i oznaka izvora sadržaja <source>. Unutar izvora označenih sa <source> smještene su pripadne datoteke video i audio sadržaja. Vrsta dokumenta, koji se prikazuje u Internet pregledniku, označava se unutar oznake <!DOCTYPE>, što uvelike olakšava pretraživanje na Web-u[2].Unutar spremnika označenog s <embeded> smještaju se vanjske korištene aplikacije ili interaktivni sadržaji (*eng. plugin*).

Unutar HTML5-a podržan je princip primanja i odlaganja (*eng. drag and drop*)[6]. Na ovaj je način moguća promjena pozicije svakog elementa koji je označen pokretnim, primjerice <img draggable="true">. HTML5 sadrži geolokacijsko pristupno sučelje[7]. Ono omogućava otkrivanje geografske pozicije korisnika aplikacije u Web pregledniku. Omogućavanje ove funkcije korisnik mora eksplicitno dozvoliti. Lokaciju izraženu u geografskim koordinatama dobivenu metodom *getLocation()/getCurrentPosition()* moguće je prikazati i na karti pomoću metode showPosition().

HTML5 podržava kreiranje para ključeva (privatni i javni), korištenih prilikom osiguravanja sigurnosti, pomoću novog elementa <keygen>[8]. Privatni se ključ pohranjuje lokalno, a javni se nalazi na poslužitelju i šalje korisniku preko mreže. Pomoću javnog ključa kreira se korisnički certifikat. Rezultat izračuna određene metode ili skripte smješta se u novi element <output>[8]. Popis unaprijed definiranih mogućnosti za ulazne kontrole specificiran je unutar elementa <datalist>[8]. Njime se definira padajući izbornik (*eng. drop down list*) predefiniranih akcija, koje je moguće odabrati. Osim novih elemenata forme, HTML5 donosi i elemente pogodne za strukturiranje dokumenta. Osim prethodno navedenih elemenata <article>, <section>, <header> i <footer> u važnije elemente ubraja se <nav> - za definiranje navigacijskih poveznica, <details> - za opis dodatnih

detalja, koji se proizvoljno mogu sakriti i <rubi> - za iznošenje napomena u programskom jeziku Ruby[9]. Dokument je moguće podijeliti u odlomke korištenjem elementa <section>, a pojedini se dijelovi teksta mogu označiti elementom <mark>[9]. Grupiranje se obavlja pomoću elementa <hgroup>. Elementom <rt> moguće je naglasiti izgovor pojedinih riječi. Korištenje proizvoljne mjerne jedinice definira se unutar elementa <meter>. Element <dialog> otvara pripadni prozor za korisničku interakciju. Samostalan sadržaj označava se elementom <figure>. Vrijeme se prati pomoću elementa <time>. Elementi <article>, <section>, <header>, <footer>, <figure>, <nav> i <aside> semantički opisuju strukturu dokumenta u Web pregledniku[9].



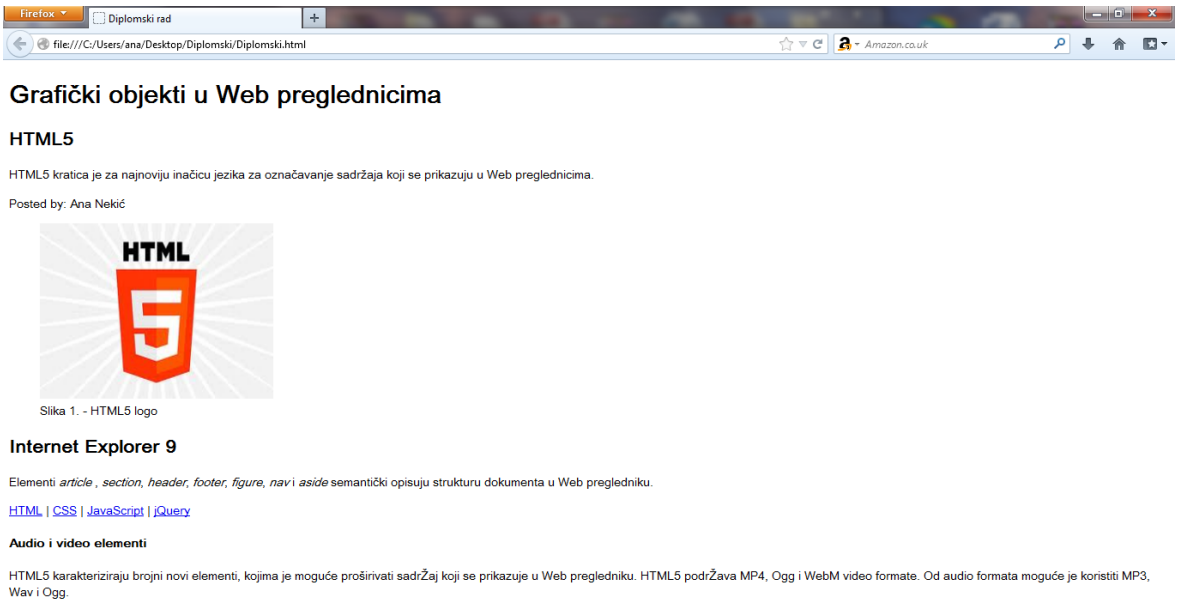

**Slika 1. Struktura HTML5 stranice[9]**

Primjer HTML5 dokumenta:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title> Diplomski rad </title>
</head>

<body>
<span style="font-family:Arial Unicode MS">
```

```
<header>
    <h1>Grafi&#269;ki objekti u Web preglednicima</h1>
    <p><time pubdate datetime="2013-06-18"></time></p>
</header>

<section>
  <h1>HTML5</h1>
<p>HTML5 kratica je za najnoviju ina&#269;icu jezika za ozna&#269;avanje
sadr&#382;aja koji se prikazuju u Web preglednicima.</p>
</section>

<footer>
  <p>Posted by: Ana Neki&#263;</p>
  <p><time pubdate datetime="2013-06-18"></time></p>
</footer>

<figure>
  <img src="html5.jpg" alt="HTML5" width="304" height="228">
  <figcaption>Slika 1. - HTML5 logo</figcaption>
</figure>

<article>
  <h1>Internet Explorer 9</h1>
  <p>Elementi  <i>article </i>, <i>section</i>, <i>header</i>,
<i>footer</i>, <i>figure</i>, <i>nav</i> i <i>aside</i> semanti&#269;ki
opisuju strukturu dokumenta u Web pregledniku. </p>
</article>

<nav>
<a href="/html/">HTML</a> |
<a href="/css/">CSS</a> |
<a href="/js/">JavaScript</a> |
<a href="/jquery/">jQuery</a>
</nav>

<aside>
  <h4>Audio i video elementi</h4>
  <p>HTML5 karakteriziraju brojni novi elementi, kojima je mogu&#263;e
pro&#353;irivati sadr&#381;aj koji se prikazuje u Web pregledniku. HTML5
podr&#381;ava MP4, Ogg i WebM video formate. Od audio formata mogu&#263;e
je koristiti MP3, Wav i Ogg. </p>
```

```
</aside>

</span>
</body>
</html>
```




**Slika 2. Izgled stranice u Web pregledniku**

HTML5 podržava SVG[10], tj. skalabilnu vektorsku grafiku *(eng. Scalabile Vector Graphics*). Skalabilna vektorska grafika koristi strukturu vektora za opis dvodimenzionalnih grafičkih elemenata, a definirana je u XML formatu. Svaki kreirani grafički element ili atribut dostupan je za animaciju i postavljanje slušatelja događaja. Važno svojstvo SVG-a je očuvanje rezolucije, tj. ne dolazi do vizualnog narušavanja prikaza prilikom povećanja ili promjene veličine prozora Web preglednika. SVG objekti posjeduju svojstvo skalabilnosti, a moguće ih je i jednostavno pretraživati, indeksirati i komprimirati. Zbog samostalnosti svakog grafičkog objekta, pomoću SVG-a moguće je jednostavno mijenjati njegovu veličinu. Ovo je glavna prednost SVG-a naspram platna. Kod platna grafički su objekti smješteni unutar prikaza, koji se iscrtava u slikovnim elementima i time je otežano mijenjanje veličine objekta, jer pritom treba iznova iscrtati cijeli prikaz.

HTML5 unosi nove formate ulaznih vrijednosti. Osim omogućavanja novih značajki, njima se doprinosi poboljšanju nadzora nad ulaznim vrijednostima i njihovom validacijom. One se definiraju pomoću : <input type="naziv_tipa" name="argument">[11]. Na ovaj je način moguće definirati boju (*eng. type color*), datum (*eng. type date*), email adresu (*eng. type email*), vrijeme i datum ovisno o vremenskoj zoni (*eng. type datetime*), lokalno vrijeme (*eng. type datetime-local*), broj telefona (*eng. type tel*), polje pretrage (*eng. type search*), url (*eng. type url*), proizvoljno vrijeme (*eng. type time*), tjedan i godinu (*eng. type week-year*), interval (*eng. range*) i brojčanu vrijednosti (*eng. number*)[11]. Za interval i brojčanu vrijednost se uz tip i naziv postavljaju i pripadna ograničenja(„min" i „max").

Podaci se u HTML5-u pohranjuju na dva načina – pohrana tijekom sjednice i lokana pohrana. Lokalnu pohranu osigurava objekt *localStorage*, a ona predstavlja lokalnu pohranu podataka na klijentu[12]. Ova je lokalna pohrana trajna. Na klijentu se obavlja i pohrana tijekom sjednice, za koju je zaslužan objekt *sessionStorage*[12]. Podaci se pohranjuju samo tijekom trajanja pojedine sjednice, nakon toga se brišu.

Novost u HTML5-u je aplikacijska priručna memorija (*eng. application cache*)[13]. Njezino je korištenje potrebno eksplicitno navesti : <html manifest="naziv.appcache">. Manifest, tj. aplikacijska priručna memorija implementirana je kao tekstualna datoteka, unutar koje su navedeni podaci koje je preglednik dužan pohraniti. Ona se sastoji od tri dijela – *CACHE MANIFEST*, *NETWORK* i *FALLBACK*[13]. Oznakom *CACHE MANIFEST* obilježavaju se datoteke, koje je potrebno pohraniti nakon prvog preuzimanja. Nakon oznake *NETWORK* navode se datoteke potrebne za uspostavljanje veze s poslužiteljem, koje se ne pohranjuju. Posljednji dio – *FALLBACK*, predstavlja rezervnu opciju u slučaju pogreške. Naime, u slučaju nedostupnosti originalne html stranice, u Web pregledniku se prikazuje stranica navedena nakon ove oznake.

HTML5 osigurava komunikaciju između klijenta i poslužitelja[14], radi ažuriranja podataka. Komunikacija između klijenta i poslužitelja je jednostrana, tj. poslužitelj šalje klijentskoj aplikaciji podatke koje je potrebno ažurirati, odnosno osvježiti. Komunikacija se odvija preko tzv. poslužiteljevih događaja, tj. obavijesti (*eng. server sent events*) koje se šalju automatski. HTML5 omogućava izvođenje skripte

na način da ona ne utječe na aktivnost stranice u Web pregledniku. Objekt *Worker* predstavlja skriptu koje se izvršava bez narušavanja rada stranice[15]. Pritom je omogućena komunikacija u vidu slanja i primanja poruka između navedene skripte i stranice koja se prikazuje u Web pregledniku. Komunikacija se prekida gašenjem *Worker* objekta, pomoću metode *Worker.terminate()*[15].

# 3. HTML5 platno

Platno (*eng. canvas*) predstavlja novi element unutar HTML5 standarda[3]. Ono osigurava skripte s rezolucijski ovisnim tzv. bitmap platnom (*eng. bitmap canvas*), koje omogućuje prikazivanje zahtjevnih vizualizacija i animacija računalne grafike u realnom vremenu, prilikom rukovanja s Web stranicom.

Grafičke elemente prikazane na platnu moguće je klasificirati u statične i dinamične. Ukoliko je priroda prikaza statična, tj. unaprijed je definiran određen prikaz, tada je navedeni prikaz ugrađen u samo platno, definirano njegovom veličinom i bitmapom. Alternativno, na platnu je predstavljena njegova rezervna opcija. Naime, prilikom korištenja platna potrebno je pružiti sadržaj, koji prikazom i svrhom odgovara konačnom vizualnom identitetu i funkciji platna dokumenta. Navedeni je sadržaj moguće postaviti kao sadržaj samog elementa platna. Na taj način osigurava se rezervna opcija prikaza samog platna. Zahtjeva li se dinamičnost platna, tada je istu potrebno implementirati unutar postojeće skripte platna. Na platnu se pritom prikazuje ugrađeni sadržaj dinamički stvaranih slika.

Prilikom prikaza ugrađenog sadržaja na platnu, moguće je usredotočiti se na tzv. prethodnike, tj. rezervni sadržaj. Pritom, iako usredotočeni element nije vidljiv, moguće je uspostaviti interakciju pomoću tipkovnice. Na ovaj način se omogućuje interakcija s tipkovnicom određenih regija platna. Interaktivnim regijama s rezervnim potencijalno usredotočenim elementima je potrebno osigurati „jedan na jedan“ preslikavanje. Interakcija pomoću miša nije omogućena navedenim pristupom.

Korištenje platna se ne preporučuje ukoliko je prisutan prihvatljiviji element. Primjerice, za prikaz grafički zahtjevnijeg naslova stranice ne koristi se platno, nego kombinacija korištenja HTML elementa *h1* (*eng. heading 1* – glavni naslov), koji je moguće stilizirati pomoću CSS-a (*Cascading Style Sheets* – jezik za opis semantike prikaza dokumenta), uz potporu tehnologije poput XBL-a (*XML Binding Language* – jezik za označavanje zasnovan na XML-u). Ukoliko platno nije potrebno koristiti, moguće je privremeno onemogućiti element platno i pripadajuću skriptu. U tom slučaju, na platnu se prikazuje njegov rezervni sadržaj.

## 3.1. Postavke platna

Veličinu platna moguće je definirati pomoću dva atributa – širine i visine, koji poprimaju cjelobrojne pozitivne vrijednosti. Unaprijed definirane vrijednosti atributa iznose 150 za visinu i 300 za širinu. Unutarnje dimenzije platna odgovaraju dimenzijama bitmapa elementa ugrađenog sadržaja, koji se prikazuje.

Platno može imati unaprijed definiran prikaz. Inicijalno, ovo nije slučaj. Stoga je ovu postavku moguće definirati unutar atributa *canvas context mode*, koji može poprimiti vrijednosti *none, direct-2d, direct-webgl, indirect i proxied*[16].

Ukoliko platno ne posjeduje unaprijed definirani prikaz, njegov bitmap nužno mora biti crne boje i potpuno proziran, a unutarnja širina i visina odgovarati njegovim vrijednostima (širini i visini samog platna). Navedene vrijednosti se prenose na CSS slikovne elemente i ažuriraju prema potrebi. U suprotnom, ukoliko je prikaz unaprijed definiran, on pruža potporu iscrtavanju, a unutarnji atributi širine i visine, kao i izgled odgovaraju elementu koji se prikazuje. Pri tome *canvas context mode*[16] poprima vrijednost *direct-2d*.

IDL atributi širine i visine trebaju odražavati pripadajuće atribute sadržaja odgovarajućeg imena i početnih postavki. IDL dolazi od Web IDL[17], tj. jezika za definiciju sučelja koja se implementiraju unutar Internet preglednika. Web IDL definira brojne značajke za omogućavanje ponašanja uobičajenih objekata skripti na različitim Web platformama.

Primjerice :

```
<div id="Naziv" class="Primjer"></div>
```

Atributi sadržaja u kodu su *id* i *class*. Navedeni atributi imaju pripadajuće IDL atribute :

```
document.getElementById('Naziv').className = 'Primjer'
```

Platno dohvaća sadržaj koji je potrebno prikazati pomoću metoda *getContex()* i *setContex().* Metodom *getContex()* vraća se objekt koji pruža potporu za iscrtavanje. Moguće je specificirati dvije vrste objekta – *2d* i *webgl*. Metoda *setContex()* postavlja element sadržaja koji je potrebno prikazati. Navedene metode se međusobno isključuju.

Odabirom *2d* prikaza odabire se objekt koji posjeduje bitmap, koji je moguće vezati na izlazni bitmap. Ovo se izvodi prilikom stvaranja objekta, a vezani ili nevezani sadržaj moguće je naknadno podesiti. Općenito, objektov bitmap služi za interakciju sa skriptom, a izlazni se bitmap prikazuje na platnu[16]. Navedeni bitmapi posjeduju identične dimenzije. Također, svaki od bitmapa odlikuju i tzv. interaktivne regije s mišem (*eng. hit regions*). Bitmap objekta posjeduje i listu neriješenih akcija sučelja. Akcije uobičajeno uključuju upute iscrtavanja određenog dijela bitmapa, kao i upute za pomicanje na karakterističnu poziciju bitmapa.

Dvodimenzionalni prikaz predstavlja površinu definiranu Kartezijevim koordinatnim sustavom, čije je ishodište točka (0,0). Ishodište je smješteno u gornji lijevi kut. Vrijednost koordinate *x* povećava se pomicanjem u desno, a koordinate *y* pomakom prema dolje. Vrijednost koordinate *x* na desnom rubu odgovara atributu širine bitmapa objekta. Identična relacija vrijedi i za vrijednost koordinate *y* na donjem rubu i visinu bitmapa objekta. Atributi širine i visine sadrže odgovarajuće vrijednosti dimenzija objektovog bitmapa, izražene u CSS slikovnim elementima.

Vrijednosti unutar korištenog koordinatnog sustava ne odgovaraju nužno dimenzijama stvarnog bitmapa, koji se interno koristi ili iscrtava na platnu. U slučaju visoke rezolucije prikaza koristi se princip „dvaju piksela po jedinici mjere“[18] (*eng. two pixel device per unit*) koordinatnog sustava, čime kvaliteta prikaza ostaje očuvana. Isti se princip naduzorkovanja (*eng. oversampling*) koristi

u slučaju antialiasinga, na način da su interno korišteni bitmapi veće rezolucije od krajnjeg prikaza na platnu.

CSS-ov atribut za označavanje boje – *currentColor* sadrži parametar odgovarajuće vrijednosti boji i računa se na dva načina. Ukoliko je riječ o unaprijed definiranom kontekstu i iscrtavanju prikaza, vrijednost parametra *currentColor* jednaka je odgovarajućoj vrijednosti boje elementa platna u trenutku prije poziva metode za iscrtavanje. U protivnom vrijednost parametra *currentColor*[16] odgovara potpuno neprozirnoj crnoj boji (*eng. fully opaque black*). Općenito tzv. *CanvasGradient* objekte karakterizira neprozirna crna boja, zbog njihovog svojstva neutralnosti. CanvasGradient objekti mogu se koristiti i na drugim platnima, stoga nije jednostavno utvrditi točnu vrijednost boje (*currentColor*) u trenutku njihovog nastajanja.

Prilikom odabira algoritma bojanja koristi se tzv. *CanvasFillRule* enumeracija. Algoritmom bojanja utvrđuje se nalazi li se određeni slikovni element unutar ili izvan područja bojanja, te treba li ga kao takvog obojiti. Prisutna su dva pravila – pravilo nenegativne vrijednosti (*eng. nonnegative rule*) i pravilo parne i neparne vrijednosti (*eng. odd-even rule*). Pravilo nenegativne vrijednosti[19] naziva se i navijajuće pravilo (*eng. winding rule*). Prema njemu, točka se nalazi izvan određenog oblika ukoliko polupravac povučen iz te točke u jednom smjeru ima isti broj presjeka s objektom kao i polupravac u drugom smjeru. Pravilo parne i neparne vrijednosti[19] definira da se točka nalazi izvan objekta ukoliko je broj presjeka polupravca povučenog iz točke prema objektu parna vrijednost.

Na stanje HTML5 elementa – platno utječu trenutno pohranjene vrijednosti unutar transformacijske matrice, matrice odsijecanja i atributa vezanih uz kompoziciju, sjenčanje, prozirnost, boje i tekstualne značajke. Za pohranu trenutnog stanja konteksta platna koristi se pripadni stog za iscrtavanje (*eng. drawing stack*). Pomoću metode *context.save()* na stog se pohranjuje kopija trenutnog stanja konteksta. Povratak na prethodno pohranjeno stanje obavlja se pomoću metode *context.restore().* Ona ponovno aktivira posljednje spremljeno stanje na stogu, na način da u matrice i atribute unosi vrijednosti koje odgovaraju stanju, koje je potrebno aktivirati. Sadržaj stoga potrebno je obrisati prilikom resetiranja.

Svaki objekt konteksta sadrži transformacijsku matricu. Prilikom stvaranja objekta ona je jednaka matrici identiteta. Pomoću metode *currentTransform*[16] dobiva se trenutno stanje transformacijske matrice. U transformacije objekta ubrajaju se translacija (*eng. translate*), skaliranje (*eng. scale*) i rotacija (*eng. rotate*). Njihovo djelovanje na transformacijsku matricu opisano je istoimenim metodama. Pomoću metode *transform(a,b,c,d,e,f)* vrijednosti unutar transformacijske matrice zamjenjuju se rezultatima njihovog umnoška s matricom čiji su redci a,b,c; d,e,f; 0,0,1.

Platno podržava definiranje proizvoljnog stila iscrtavanja (*eng. Drawing Style*) pomoću istoimenog objekta. Opseg djelovanja stila za iscrtavanje definira se pomoću argumenta koji se predaje konstruktoru *DrawingStyle*[16]. Alternativno, koristi se definicija stila iscrtavanja prisutna u globalnoj okolini JavaScript. Osim linija, moguće je odabrati proizvoljni stil teksta, koji se prikazuje unutar platna. IDL atribut proizvoljnog *font* objekta se prevodi na jednak način kao i CSS-ovo svojstvo postavke fonta. Navedeni se font potom pridjeljuje željenom kontekstu, a njegova veličina prevodi u CSS slikovne elemente. Osim sistemski podržanih fontova moguće je odabrati i vlastiti font i učitati ga pomoću *FontLoader* objekta. On posjeduje svojstvo robusnosti, tj. ukoliko se koristi bez prethodnog ispravnog učitavanja, ne dolazi do dojave greške već se umjesto njega koristi neki od fontova valjanje CSS specifikacije.

Unutar sučelja *CanvasPathMethods* nalaze se metode za crtanje matematičkih funkcija i objekata. Svaka od navedenih metoda podržava kreiranje tzv. podstaze (*eng. subpath*). Podstaza predstavlja listu točaka povezanih pravcem ili krivuljom. Svaka podstaza može biti zatvorena ili otvorena, što je definirano pripadnom zastavicom. Za zatvorenu podstazu je karakteristična povezanost prve i zadnje točke unutar liste pomoću ravne linije. Nova podstaza nastaje metodom *path.moveTo(x,y)*, a zatvara je metoda *path.closePath()*. Pritom nastaje i nova podstaza, čija je početna točka jednaka posljednjoj točki zatvorene podstaze. Nova se točka podstazi dodaje ovisno o željenom obliku. Moguće je na taj način kreirati kvadratnu Bezierovu krivulju (*eng. path.quadraticCurveTo(cpx, cpy, x,y)*), kubnu Bezierovu krivulju (*eng. path.bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x,y)*), kružni luk (*eng. path.arcTo(x1,y1,x2,y2, radiusX[radiusY, rotation])*), pravokutnik (*eng. path.rect(x,y,w,h)*) ili elipsu(*eng. path.ellipse(x,y,radiusX, radiusY, rotation,*

*startAngle, endAngle, anticlockwise)*)[16]. Prije dodavanja točke u podstazu, na nju djeluje transformacijska matrica. Prilikom iscrtavanja, za svaku se koordinatu provjerava postojanje podstaze, koja je sadrži. Navedene metode djeluju isključivo na posljednju podstazu objekta.

### 3.1.1. Kompozicija iscrtavanja

Procesom iscrtavanja upravlja se pomoću glavnih atributa – *globalAlpha* i *globalCompositeOperation*. Atribut *globalAlpha* pridjeljuje vrijednosti alfa parametra dijelovima prikaza, prije njihovog unosa u objektov bitmap (*eng. scratch bitmap*). Alfa parametar utječe na prozirnost, a njegova je vrijednost u intervalu od 0 (potpuno prozirno) do 1 (potpuno neprozirno). Prilikom stvaranja konteksta iscrtavanja njegova je vrijednost potpuno neprozirna.

Smještajem dijelova unutar prikaza objektovog bitmapa upravlja atribut *globalCompositeOperation*[16]. Ono se odvija nakon dodjele alfa vrijednosti i primjene transformacijske matrice. Prilikom postavke smještaja razlikuje se izvorna slika (*eng. source image*) i odredišna slika - trenutno stanje objektovog bitmapa (*eng. destination image*). Pretpostavljena vrijednost atributa *globalCompositeOperation* je „*source - over*“, tj. izvorna slika se smješta na područja gdje je neprozirna, a ostatak prekriva odredišna slika. „*Destination - over*“ odvija se na isti način, samo su uloge izvorne i odredišne slike zamijenjene. U slučaju „*source - atop*“, izvorna se slika smješta na područje gdje su obje – i izvorišna i odredišna slika neprozirne. Na područja gdje je izvorna slika prozirna, a odredišna neprozirna smješta se odredišna slika. Ostatak se ostavlja prozirnim. Obrnuti postupak, tj. zamjena uloga izvorne i odredišne slike karakterizira „*destination - atop*“. Pomoću „*source - in*“ izvorna se slika smješta samo na područja gdje su i izvorna i odredišna slika neprozirne, a ostatak se ostavlja prozirnim. Njoj suprotna vrijednost je „*destination - in*“. Ukoliko je odabran „*source - out*“ izvorna se slika smješta u područje, gdje je izvorna slika neprozirna, a odredišna prozirna. Obrnuti postupak – zamjena uloga odredišne i izvorne slike, odlika je „*destination - out*“. Opcija „*lighter*“ predstavlja sumu izvorne i odredišne slike, koja se kao takva smješta u prostor bitmapa. Prikaz dobiven operacijom

„*isključivo – ili*“ između izvorne i odredišne slike dobiva se opcijom „*xor*“. Ako je odabrana opcija „*copy*“ u prostor se smješta isključivo izvorna lika. U svakoj od navedenih operacija djeluje se isključivo na određenu regiju objektovog bitmapa, ne na cjelokupni bitmap.

### 3.1.2. Sjenčanje

HTML5 element platno podržava sjenčanje, čiju boju određuje *shadowColor* atribut. Prilikom stvaranja konteksta sjenčanje je inicijalno prozirno – crne boje. Boja sjenčanja prenosi se kao CSS vrijednost (kao i boja). Sjenčanje djeluje na prostor omeđen vrijednostima unutar atributa *shadowOffsetX* i *shadowOffsetY*. Vrijednosti navedenih atributa izražene su u jedinicama korištenog koordinatnog sustava i na njih ne utječe transformacijska matrica.

Djelovanje efekta zamućenja (*eng. blur*) moguće je proizvoljno postaviti, definiranjem parametra *shadowBlur*. Dio prikaza bit će osjenčan ukoliko je alfa parametar boje definirane unutar *shadowColor* pozitivne vrijednosti i ako se isto odnosi i na jedan od parametara *shadowOffsetX*, *shadowOffsetY* i *shadowBlur*[16].

### 3.1.3. Gustoća slikovnih elemenata i ispravljanje boje

Gustoću slikovnih elemenata, tj. rezoluciju objektovog bitmapa, kojeg koristi platno, moguće je proizvoljno odabrati. No, početno odabrana rezolucija bitmapa treba se trajno održavati. Općenito se preporučuje korištenje gustoće slikovnih elemenata, koja odgovara vrijednostima gustoće slikovnih elemenata zaslona. U idealnom slučaju broj slikovnih elemenata zaslona, kojima je predstavljen svaki CSS slikovni element, bio bi višekratnik broja dva. Svi bitmapi stvoreni za pružanje podrške obavljanju specifičnog zadatka moraju imati jednaku gustoću slikovnih elemenata, kao i *CanvasRendering2D* objekti. Rezolucija platna (*eng. canvasResolution*) definira se u sklopu *Screen*[16] objekta *canvasResolution*.

Ispravljanje boje obavlja se prilikom iscrtavanja bitmapa i prilikom iscrtavanja prikaza na platnu iz navedenog bitmapa. Prilikom iscrtavanja slike u bitmap obavlja se Gama ispravljanje (*eng. Gamma correction*) i pretvaranje parametara boje u pripadne vrijednosti boje bitmapa. Gama ispravljanje je nelinearno preslikavanje parametara osvijetljenosti u statičkim ili dinamičkim vizualnim objektima. Gama ispravljanje i usklađivanje parametara mora osigurati da boje prisutne kod iscrtavanja direktno iz *img* elementa odgovaraju bojama na platnu nakon iscrtavanja istog.

Pomoću ispravljanja boja potrebno je obuhvatiti cjelokupni obujam boja dostupan unutar CSS-a. Također, potrebno je osigurati što bolju usklađenost stvarne slike i njezinog prikaza na platnu. Stoga slikovni elemenati obojeni metodom *toDataURLHD()* moraju odgovarati dobivenim vrijednostima metode *getImageDataHD().*

## 3.2. Interaktivne regije

Interaktivne regije s mišem[20] (*eng. hit regions*) predstavljaju jednu od važnijih značajki HTML5 platna. One pružaju način za definiranje dijelova HTML5 platna i pridjeljivanje „id:property“ svojstva istima. Klikom miša ili drugim načinom interakcije s mišem moguće je usporediti „*property regije na event object-u“* s id svojstvom hit regije[20]. Ovim pristupom moguće je utvrditi interakciju bez upotrebe koordinata miša. Također, na intuitivan način moguće je detektirati klik na pojedini objekt složene geometrije.

Interaktivnu regiju s mišem moguće je dodati naredbom *context.addHitRegion({'id': 'button1', 'cursor': 'pointer'})*. Metoda kao argument prima objekt s navedenim svojstvima:

- *path* – *Path* objekt platna određuje oblik hit regije.
- *id* – Identifikacija karakteristične *hit* regije. Nakon interakcije s mišem, svojstvo *event.region* postavlja se na identifikacijsku oznaku regije na koju je interakcija primijenjena.
- *parentID* – Identifikacijska oznaka roditeljske hit regije.

- *cursor* – Postavljanje izgleda pokazivača miša na platnu za pojedinu regiju.
- *control* – Kontrola kojem elementu platna (najčešće gumb) će biti proslijeđena interakcija s mišem.
- *label* – Ukoliko je izostavljena kontrola, moguće je tekstualno označiti pojedinu regiju. Tekstualna se oznaka ne treba prikazati na platnu.
- *role* – Ukoliko nije definirana kontrola, pomoću uloge moguće je pridijeliti ulogu pojedinoj regiji.

Primjer dodavanja *hit regije*[20]:

```
<script>
    context.beginPath();
    context.rect(10,10,100,100);
    context.fill();
    context.addHitRegion({'id': 'Regija 1', 'cursor': 'pointer'});

    context.beginPath();
    context.rect(120,10,100,100);
    context.fill();
    context.addHitRegion({'id': 'Regija 2', 'cursor': 'pointer'});

    canvas.onclick = function (event)
    {
        if (event.region) {
            alert('Odabrali ste regiju pod nazivom ' + event.region);
        }
    }
<script>
```

Trenutno *hit regije* nisu podržane od strane Web preglednika. No, moguće ih je iskoristiti za detekciju klika miša prilikom geometrijskih izračuna, DIV oznaka na platnu (*eng. div tags*) i ponovne reprodukcije određene staze (*engl. path*) pomoću metode *isPointInPath()*.

# 4. WebGL

WebGL predstavlja aplikacijsko programsko sučelje za potporu prikaza 3d grafike na Internetskim stranicama[21]. Nastao je iz OpenGL-a ES 2.0 i pruža sličan opseg funkcionalnosti u kontekstu HTML-a. Naime, WebGL je razvijen za podršku prikaza grafike u sklopu HTML5 elementa platna. Grafički sadržaj prikazan na HTML5-u moguće je razvijati na dva načina - 2d (*eng. CanvasRenderingContext*) i WebGL (*eng. WebGLRenderingContext*).

Glavnu prednost WebGL aplikacijskog programskog sučelja predstavlja njegova neposrednost, koja ga uvelike razlikuje od sličnih Web sučelja. S obzirom na različitu primjenjivost 3d grafike, WebGL je pogodan za razvijatelje, ponajviše zbog pristupa. WebGL-ov pristup pruža fleksibilne primitive grafičkih elemenata, koje su prilagodljive raznolikim aplikacijama, ovisno o prirodi njihove svrhe. WebGL-ove knjižnice (*eng. library*) pružaju veliki izvor različitih funkcija, specifično prilagođenih pojedinim područjima grafike, te predstavljaju intuitivan sloj za korisnika, pojednostavljujući i ubrzavajući njegov rad. Ovo je prvenstveno odnosi na poznavatelje OpenGLa ES 2.0, iz kojeg je nastao WebGL.

## 4.1. Sadržaj i spremnik za iscrtavanje

Inicijalno je potrebno definirati sadržaj koji se prikazuje pomoću WebGL-a (*eng. WebGLRenderingContex*) i povezati ga s HTML5 elementom – platnom. Njegovi se atributi postavljaju u sklopu objekta *WebGLContexAttributes*[21].

Spremnik za iscrtavanje (*eng. drawing buffer*) predstavlja spremnik u kojem se iskazuju pozivi aplikacijskog programskog sučelja WebGL-a. On se definira zajedno s objektom *WebGLRenderingContex*.

Spremnik za iscrtavanje sastoji se od tri spremnika – spremnika boje (*eng. color*), spremnika dubine (*eng. depth*) i spremnika maskiranja (*eng. stencil*).

Spremnik boje[21] služi za pohranu RGBA parametara. RGBA je naziv za vrijednosti parametara crvene, zelene i plave boje, te alfa parametra. Alfa

parametar označava prozirnost. Ako je alfa parametar jednak nuli, riječ je o potpunoj prozirnosti (nevidljivosti). Najveća vrijednost alfa parametra je 255 – neprozirnost karakteristična za digitalan format. Spremnik boje zauzima 32 bita, odnosno 8 bitova po svakoj RGBA komponenti i inicijalno je prisutan.

U spremnik dubine[21] pohranjuje se koordinata dubine. Naime, iscrtavanje je kod WebGL-a cjevovodni proces. Početno se pomoću programa za sjenčanje vrhova (*eng. vertex shader*) učitavaju pozicije vrhova. Potom dolazi do linearne interpolacije između vrhova, kako bi se definirali dijelovi koje je potrebno obojiti (pomoću RGBA parametara u spremniku boje). Nakon toga nastupa iscrtavanje, za koje je osobito važna komponenta dubine pohranjena u spremniku dubine. Komponenta dubine povezana je sa z-komponentom. Naime, u WebGL-u su pretpostavljene vrijednosti dubine u rasponu 0 – 1, pri čemu 0 predstavlja najbližu, a 1 najdalju točku od promatrača. Prilikom iscrtavanja, dio s većom vrijednosti komponente dubine zamjenjuje se dijelom s manjom vrijednosti komponente dubine. Provjera vrijednosti se vrši prilikom iscrtavanja svakog od dijelova prikaza. Na ovaj se način iscrtava ispravna slika. Spremnik dubine zauzima 16 bitova.

Spremnik maskiranja[21] koristi se za ograničavanje područja na kojem se obavlja prikazivanje, tj. maskiranje. Zauzima 8 bitova i predstavlja prvenstveno poboljšanje sklopovlja. Kombinacijom dubine i maskiranja nastaju zanimljivi efekti poput postavljanja sjena, osvjetljavanja granice između složenih primitiva ili uokvirivanja dijelova prikaza. Podaci iz spremnika maskiranja najčešće se koriste za sjenčanje u 3D aplikacijama. *WebGLRenderingContext*, tj. sadržaj prikaza, otporan je na korištenje različitih kombinacija atributa. Ukoliko se koristi nedopuštena kombinacija, ona će se ignorirati i neće doći do dojave greške.

U ključne se atribute ubrajaju *alpha, premultipliedAlpha i preserveDrawingBuffer*. WebGL izlaže sadržaj spremnika za iscrtavanje pregledniku HTML stranice neposredno prije početka učitavanja. Ovo se obavlja jedino ako je nastupila promjena u spremniku za iscrtavanje. Prije izlaganja sadržaja spremnika za iscrtavanje, nužno je osigurati izvođenje svih predviđenih operacija. Nakon osvježavanja HTML stranice u pregledniku, potrebno je inicijalizirati vrijednosti spremnika za iscrtavanje. Pritom se vrijednosti parametara RGBA unutar spremnika boje postavljaju na (0,0,0,0), spremnik dubine na 1.0, a spremnik

maskiranja na 0. Inicijalizaciju spremnika moguće je izbjeći postavljanjem zastavice atributa *preserveDrawingBuffer*. Na ovaj način sadržaj spremnika za iscrtavanje ostaje očuvan dok ga korisnik ručno ne izmijeni. Atribut *premultipliedAlpha* uzrokuje množenje parametra alfa s parametrima boje. Obično se koristi za pojednostavljivanje kasnijih operacija množenja. Parametar *premultipliedAlpha* osobito utječe na funkcije *toDataURL()* i *drawImage()*. Naime, ako format zapisa slike koju je potrebno prikazati na HTML5 platnu ne specificira korištenje modificiranog parametra alfa, a unutar objekta *WebGLContext* je postavljena zastavica parametra *premultipliedAlpha*, potrebno je podijeliti vrijednosti parametara boje slikovnog elementa s vrijednosti parametra alfa. Prilikom rukovanja s teksturama pomoću sučelja funkcije *texImage2D()*, ovisno o stanju atributa *premultipliedAlpha* i parametra slikovnog elementa *UNPACK_PREMULTIPLY_ALPHA_WEBGL*, potrebno je modificirati izvorne zapise slikovnih elemenata množenjem ili dijeljenjem parametara boje s parametrom alfa. WebGL nastoji poštivati vrijednosti zadane unutar spremnika boje, dubine i maskiranja.

Veličinu spremnika za iscrtavanje određuju visina i širina HTML5 platna. No, spremnik jednakih dimenzija kao i platno nije uvijek slučaj. Ukoliko inicijalno ili uslijed naknadne promjene nije moguće kreirati spremnik za iscrtavanje dovoljne veličine, spremnik manjih dimenzija postaje prihvatljiva opcija. Ispravne dimenzije spremnika za iscrtavanje pohranjuju se u atributima *drawingBufferWidth i drawingBufferSize*.

## 4.2. DOM sučelje

Resursi se u WebGL-u predstavljaju DOM objektima. DOM[22] (*Document Object Model*) naziv je aplikacijskog programskog sučelja za HTML i strukturirane XML dokumente. DOM definira logičku strukturu dokumenata i način na koji im je moguće pristupiti i izmjenjivati ih. DOM specifikacija pojam *dokument* obuhvaća u širem smislu značenja. Naime, pomoću programskog jezika XML moguće je pohraniti različite vrste podataka u mnogim sustavima, čime pojam *dokument*

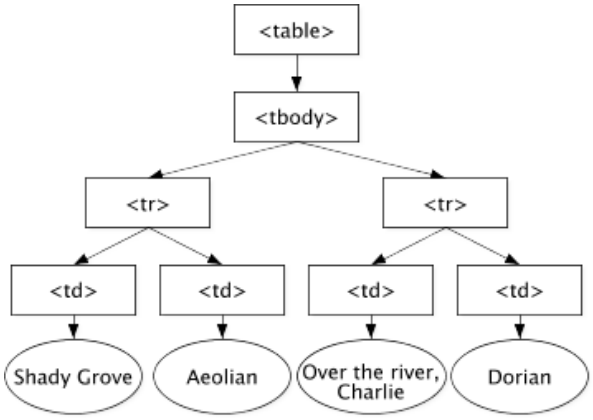
dobiva drugačiji značaj. Pomoću DOM sučelja moguće je efikasno upravljati navedenim različitim tipovima podataka, pohranjenim unutar dokumenta.

DOM sučelje omogućuje stvaranje novih dokumenata i definiciju njihove strukture, kao i jednostavno rukovođenje njihovim elementima. Modifikacije ili uklanjanje dijelova sadržaja moguće je intuitivno realizirati. Svaki dio HTML ili XML dokumenta u potpunosti je dostupan i podložan promjenama. Iznimka su pojedini vanjski i unutarnji poddijelovi XML dokumenta, prema kojima pristup DOM sučeljem nije još omogućen.

DOM sučelje temelji se na objektnoj strukturi koji nalikuje strukturi dokumenta modela. Primjerice, tablica iz XHMTL-a :

```
<table>
  <tbody>
    <tr>
      <td>Shady Grove</td>
      <td>Aeolian</td>
    </tr>
    <tr>
      <td>Over the River, Charlie</td>
      <td>Dorian</td>
    </tr>
  </tbody>
</table>
```

Istu tablicu pomoću DOM strukture prikazuje Slika 3.




**Slika 3. Grafički prikaz DOM strukture[22]**

W3C standardom definirana je otvorena platforma za razvijanje Web aplikacija. Prema W3C specifikaciji glavna uloga DOM-a jest pružiti standardno programsko sučelje, koje se može koristiti na različitim programskim okolinama i aplikacijama. DOM sučelje moguće je koristiti s bilo kojim programskim jezikom. OMG IDL (*eng. Object Managment Group Interactive Data Language*) definira preciznu, jezično neovisnu specifikaciju DOM sučelja. Također, postoje i poveznice s programskim jezikom Java, temeljene na JavaScriptu. Uslijed ograničenja uzrokovanih samom prirodom poveznica, Java ne dozvoljava korištenje sučelja, koja sadržavaju atribute, iako je isto omogućeno DOM sučeljem.

Sredstva koja koristi WebGL predstavljena su kao DOM objekti. Svaki je objekt izveden iz sučelja *WebGLObjects*. Trenutno je podržana uporaba tekstura, spremnika, grafičkih spremnika (*eng. framebuffer*), programa za sjenčanje, spremnika za prikazivanje (*eng. renderbuffers*) i drugih programa. U sklopu sučelja *WebGLRenderingContext* nalaze se metode za stvaranje *WebGLObject* objekta za svaku od navedenih vrsta sredstava. Pri tome su podaci iz pripadajućih grafičkih knjižnica u potpunosti dostupni za proizvoljno upravljanje. Također, osigurava se postojanje navedenih objekata tijekom cijelog perioda korištenja *WebGLObject* objekata. Općenito, DOM objekt postoji dok je valjana poveznica na njega ili dok je povezan pomoću pripadajućih grafičkih knjižnica. U slučaju da niti jedan od navedenih uvjeta nije ispunjen, objekt je moguće jednostavno ukloniti pripadajućom funkcijom za brisanje. Ukoliko je resurs koji je potrebno ukloniti objavljen, brisanje je potrebno eksplicitno definirati.

### 4.3. Glavne razlike OpenGL-a i WebGL-a

Spremnik aplikacijskog sučelja WebGL-a (engl. buffer) moguće je vezati uz strukture *ARRAY_BUFFER* ili *ELEMENT_ARRAY_BUFFER*, tj. unutar spremnika moguće je pohraniti vrhove ili njihove indekse, no ne oboje istovremeno. Navedeni spremnik moguće je samo jednom inicijalizirati pomoću ulaznog argumenta, koji mu se predaje. Također, u sklopu WebGL-a nema podrške za rukovanje s podacima od strane klijenta[21].

Ukoliko je unutar spremnika pohranjen niz atributa vrhova (*engl. vertex attribute*), on će biti korišten u sklopu funkcija *drawArrays* i *drawElement*. Prije iscrtavanja, funkcije će dodatno provjeriti nalazi li se svaki od vrhova iz niza unutar skupa dopuštenih vrijednosti definiranih spremnikom, koji je povezan s pohranjenim nizom, te izazvati dojavu o pogreški. Izostanak pogreške prilikom izvođenja funkcija *drawArray* i *drawElement* javlja se ukoliko se pohranjeni niz atributa vrhova ne koristi unutar programa. Tada je izvođenje ovih funkcija neovisno o vrijednostima unutar vezanog spremnika. Atribut vrhova je naziv za vrijednost, koja se za taj vrh prosljeđuje na program za sjenčanje vrhova (engl. vertex shader). Programsko sučelje WebGL-a podržava rast vrijednosti atributa vrhova do 255 bajtova, nakon čega dojavljuje pogrešku.

WebGL uvodi pristupnu točku spremniku (šablone) maskiranja dubine (*eng. DEPTH_STENCIL_ATTACHMENT framebuffer*) i unutarnji format maskiranju dubine spremnika za iscrtavanje (*eng. DEPTH_STENCIL renderbuffer*)[21]. Na ovaj način moguće je povezati spremnike dubine i maskiranja s grafičkim spremnikom. Dubina prikaza predstavlja prostor između granične udaljene ravnine i granične bliže ravnine. Pri tome nije dozvoljeno preslikavanje bliže ravnine (*eng. zNear*) na vrijednost veću od definirane vrijednosti udaljene ravnine (*eng. zFar*), što rezultira dojavom pogreške.

Rukovanje slikovnim elementima izvan raspona definiranog unutar grafičkog spremnika predstavlja unaprjeđenje funkcionalnosti. Funkcije koje čitaju vrijednosti iz grafičkog spremnika – *copyTexImage2D, copyTexSubImage2D i readPixels*[21] u slučaju pojave vrijednosti koja se ne nalazi unutar zadanih ograničenja, istoj postavljaju RGBA vrijednost na (0,0,0,0). Poboljšanje WebGL-u donose i definirani dodatni parametri pohrane slikovnih elemenata. Pomoću *boolean* parametra *UNPACK_FLIP_Y_WEBGL* moguće je tijekom naknadnih poziva funkcija *texImage2D()* i *texSubImage2D()* zrcaliti izvorne vrijednosti s obzirom na vertikalu (*y-os*), na način da se prvi prenosi posljednji red. Spomenute funkcije služe za rukovanje s teksturama. WebGL ne podržava korištenje sažetih tekstura. Inicijalno parametar nije postavljen, a moguće ga je postaviti pomoću proizvoljne numeričke vrijednosti, osim nule. *UNPACK_PREMULTIPLY_ALPHA_WEBGL* utječe na promjenu vrijednosti parametara boje uslijed prijenosa izvornih podataka, uzrokovano njihovim množenjem sa parametrom alfa, koji utječe na prozirnost.

Navedeni *boolean* parametar početno nije postavljen, a aktivira se na isti način kao i *UNPACK_FLIP_Y_WEBGL. UNPACK_COLORSPACE_CONVERSION_WEBGL* uzrokuje pretvorbu postavki boje Internet preglednika, prilikom djelovanja funkcija *texImage2D()* i *texSubImage2D()* na slikovni element HTML-a (*eng. HTML Image Element*). Moguće je specificirati pretvorbe za pojedini preglednik i zapis slike. Parametar je *longint*, a početna vrijednost jednaka je *BROWSER_DEFAULT_WEBGL*.

Općenito se kod WebGL-a prilikom sjenčanja ne dozvoljava postavljanje konstante boje i konstante alfa parametra kao početne i konačne vrijednosti. Kod OpenGL-a moguće je obavijestiti aplikaciju o dodatnim načinima formatiranja i kombinacijama koje je moguće proslijediti funkciji *ReadPixel()*, uz univerzalan par *RGBA/Unsigned_Byte*, pomoću parametara funkcija *IMPLEMENTATION_COLOR_READ_FORMAT i IMPLEMENTATION_COLOR_READ_TYPE*[21]. Ovo je uklonjeno u WebGL-u. Umjesto toga, slikovni elementi iz trenutnog stanja grafičkog spremnika mogu se učitati u novi *ArrayBufferView* objekt.

Za razliku od OpenGL-a, kod WebGL-a nisu dozvoljene aritmetičke vrijednosti sa čvrstom točkom (*eng. fixed point value – GL_FIXED*). WebGL također ograničava gniježđenje struktura unutar programa za sjenčanje. Do pojave gniježđenja dolazi kada je određeno polje unutar strukture definirano drugom strukturom. Pri tome korištenje ugrađenih struktura nije dozvoljeno. WebGL podržava gniježđenje do najviše četiri razine. Općenito, postoji ograničenje na riječi korištene u programskom kodu koje koristi WebGL na 256.

# 5. Biblioteka Three

Biblioteka Three namijenjena je podršci prilikom izrade aplikacija koje koriste HTML5 element – platno. Napisana je u programskom jeziku JavaScript, te uvelike olakšava implementaciju grafičkih aplikacija. Ona osigurava izvođenje osnovnih funkcija potrebnih za iscrtavanje, kako bi se razvijatelj mogao usredotočiti na specifične implementacijske probleme. Three biblioteka[23] pojednostavljuje korištenje WebGL metoda, na koje se referencira.

Korištenje Three biblioteke potrebno je definirati unutar glavne HTML stranice:

```
<script type="text/javascript" src="js/three.min.js"></script>
```

## 5.1. Postavke scene

Omogućavanje vizualnog prikaza općenito uključuje scenu, kameru i iscrtavanje (*eng. renderer*). Biblioteka Three osigurava specifičnu podršku za navedene uvjete vizualnog prikaza, koji se intuitivno dodaju HTML5-ovom elementu platno. Inicijalno je nužno definirati scenu:

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

### 5.1.1. Kamera

Unutar Three biblioteke postoje dvije vrste kamera – perspekcijska i ortografska[24]. Osnova obje kamere jest apstraktna osnovna klasa *Camera*. Općenito ju nasljeđuju sve daljnje dodatno implementirane klase kamera. Njezina glavna svojstva uključuju projekcijsku matricu (*eng. Projection Matrix*) i matricu njezinog inverza (*eng. Projection Matrix Inverse*), te inverznu matricu svijeta (*eng. Martix World Inverse*). Inverzna matrica svijeta sadrži transformaciju prilagođenu kameri. Kamera se postavlja pomoću metode *Camera.lookAt(vector)*. Ulazni

parametar je tipa vektor i njime su definirane koordinate unutar globalnog prostora, na koje je kamera usmjerena.

Deklaracija perspekcijske kamere obavlja se naredbom:

```
var camera = new THREE.PerspectiveCamera( vidno polje, aspect ratio, najbliža ravnina odsjecanja, najdalja ravnina odsjecanja );
```

Vidno polje (*eng. field of view*) predstavlja opseg vidljivog svijeta, ovisno o rezoluciji. *Aspect ratio* je univerzalan pojam za označavanje omjera visine prema širini prozora prikaza. Prostorni odnosi unutar prikaza definirani su najbližom i najdaljom ravninom odsjecanja. Na prikazu su prisutni isključivo objekti, čija se udaljenost od kamere nalazi u intervalu omeđenom s prethodne dvije karakteristične ravnine.

Ortografska se kamera definira pomoću naredbe:

```
var camera = new THREE.OrthographicCamera( lijeva, desna, gornja, donja, najbliža, najdalja ravnina odsjecanja );
```

Ulazni parametri konstruktora predstavljaju ravnine, kojima je omeđen prikaz. Navedene je parametre moguće naknadno ažurirati. Nakon obavljene izmjene, ista se pohranjuje pomoću metode *OrtographicCamera. updateProjectionMatrix()*[25].

### 5.1.2. Iscrtavanje

Biblioteka Three podržava više načina iscrtavanja prikaza, što je osobito pogodno kod starijih inačica pretraživača. Najčešće se upotrebljava upravo *WebGLRenderer*.

```
var renderer = new THREE.WebGLRenderer();

renderer.setSize( window.innerWidth, window.innerHeight );
```

Prilikom deklaracije instance klase iscrtavača (*eng. renderer*) potrebno je definirati veličinu iscrtanog prikaza[26]. Općenito dimenzije širine i visine prikaza odgovaraju dimenzijama pripadajućeg prozora, no to nije pravilo. U slučaju grafički zahtjevnije

aplikacije dimenzije prikaza koje se koriste manje su u odnosu na dimenzije prozora unutar kojeg se pojavljuje prikaz. Pritom konačan prikaz dimenzijama odgovara prozoru, no smanjene je rezolucije.

Podržane *CanvasRenderer* i *WebGLRendererTarget* odlikuje zajedničko svojstvo – korisnički definirani postupci, koji se opisuju pripadajućim funkcijama. Pozivom metode nastupa iscrtavanje.

Konstruktor *WebGLRenderer* sadrži nekoliko parametara, čije je korištenje opcionalno. Pripadni parametri su platno, preciznost, alfa, preračunata alfa, antialias, uzorak, postavljanje spremnika za iscrtavanje, parametri boje i afa vrijednosti unutar spremnika boje, modificirani parametar alfa i maksimalno osvjetljenje. Platno (*eng. canvas*) predstavlja poveznicu s HTML elementom na kojem se odvija iscrtavanje. Parametar preciznosti (*eng. precision*) odnosi se na preciznost programa za sjenčanje koji se koristi. Ovaj parametar može poprimiti visoku (*eng. highp*), srednju (*eng. mediump*) i nisku (*eng. lowp*) vrijednost[24]. Alfa parametar, kao i modificirani alfa parametar (*eng. premultipliedAlpha*) boolean su tipa i inicijalno postavljeni (*eng. thrue*). Antialiasing (*eng. antialias*) početno ima vrijednost *false* i potrebno ga je naknadno aktivirati. Parametar uzorka (*eng. stencil*) također je *boolean* tipa i početno je postavljen, za razliku od parametra očuvanja spremnika za iscrtavanje (*eng. preserveDrawingbuffer*), čija je početna vrijednost *false*. Parametri kojima se boja i alfa vrijednost postavljaju na nulu (*eng. clearColor* i *eng. clearAlpha*) početno su postavljeni na 0x000000 (float), odnosno 0 (integer)[24]. Parametar najvećeg osvjetljenja (*eng. maxLights*) cjelobrojnog je tipa i početno ima vrijednost 4. Ispitivanja su pokazala da se *CanvasRenderer* izvodi brže od *WebGLRenderer-a*[24]. Ukoliko se unutar aplikacije nalazi program za sjenčanje nužno je koristiti *WebGLShaders*.

Pripadne metode klase *WebGLRenderer* uključuju *getContex()* – za vraćanje pripadnog WebGL konteksta, *supportVectorTextures()* – provjera jesu li podržane vektorske teksture, *setSize(width, height)* – za postavljanje veličine, *setViewPort(x,y, width, height)* – za definiranje dijela koji se iscrtava, *setScissor(x,y,width, height)* i *enableScissorTest()* – za odjeljivanje dijela prikaza, odnosno iscrtavanje isključivo u odijeljenom dijelu[24]. Za postavljanje boje i prozirnosti koriste se *setClearColor(color, alpha*), a za dohvaćanje pripadnih

vrijednosti *getClearColor() i getClearAlpha().* Brisanje vrijednosti spremnika za iscrtavanje obavlja se metodom *clear( color, depth, stencil ).*

Podržano je korištenje vanjskih predprocesirućih, odnosno postprocesirajućih dodataka, koje se definira metodama *addPostPlugin(plugin) i addPreePlugin(plugin).* Mapa sjenčanja se tijekom izvođenja ažurira ovisno o parametrima scene i kamere pomoću metoda *updateShadowMap(scene, camera).* Iscrtavanje preko podataka iz pripadnog spremnika može se odvijati neposredno pomoću metode *renderBufferImmediate()* ili direktno metodom *renderBufferDirect(camera, lights, fog, material, geometryGroup, object)*. Cjelokupna se scena iscrtava pomoću metode *render(scene, camera, render, target, forceClear)*[24]*.* Posljednji u nizu ulaznih parametara – *forceClear* označava brisanje svih prethodnih sadržaja prikazanih na platnu prije iscrtavanja.

Objekti unutar scene postavljaju se unutar metode *initObject(scene)*, a pripadni materijali unutar metode *initMaterial(material, lights, fog, object).* Za odbacivanje geometrije koristi se metoda *setFaceCulling(cullFace, frontFace).* Prvi parametar – *cullFace* označava prostornu određenost i poprima vrijednosti *back, front, front_and_back i false*. Parametrom *frontFace* određuje se usmjerenost, a moguće vrijednosti su u smjeru kazaljke na satu - *cw* (*eng. clockwise*) i suprotno od smjera kazaljke na satu - *ccw* (*eng. counterclockwise*). Pripadni testovi dubine ostvaruju se pomoću metoda *setDepthTest(depthTest) i setDepthWrite(depthWrite)*. Prilikom iscrtavanja moguće je koristiti miješanje grafičkih atributa (boja i oblika), prema proizvoljnoj jednadžbi, pomoću metode *setBlending(blending, blendEquation, blendSrc, blendDst)*[24]*.* Proizvoljna tekstura postavlja se naredbom *setTexture(texture, slot),* pri čemu se definira područje postavljanja teksture unutar scene.

## 5.2. Geometrija

*Geometry* je osnovna klasa za pohranu definiciju geometrije trodimenzionalnih oblika.

```
var geometry = new THREE.Geometry();
```

Osim identifikacije (*eng. id*) i naziva, svojstva uključuju i niz koordinata točaka, koje pripadaju određenom objektu. Podržano je i obavještavanje o nužnosti ažuriranja podataka pomoću *Geometry.verticesNeedUpdate* zastavice. Svojstva uključuju i specifične nizove u kojima se pohranjuju boje, normale i likovi, te pripadne zastavice za ažuriranje - *Geometry.colorsNeedUpdate, Geometry.normalsNeedUpdate i Geometry.facesNeedUpdate*[24].

Moguće je definirati površinske likove – trokut i pravokutnik.

Trokut (Face3) se definira naredbom:

```
var face = new THREE.Face3( a, b, c, normal, color, materialIndex);
```

Ulazni parametri uključuju koordinate triju vrhova, nizove normala i boje svake točke unutar trokuta, te identifikacijski indeks materijala.

Iste ulazne parametre, uz dodatnu koordinatu vrha, koristi i metoda za definiciju pravokutnika (Face4):

```
var face = new THREE.Face3( a, b, c,d,normal, color, materialIndex);
```

Korištenjem naredbe *clone()* moguće je stvoriti kopiju svakog od navedenih likova.

Za svaki pojedini lik, odnosno vrh lika, moguće je definirati nekoliko UV slojeva (*eng. UV layer*). UV slojevi koriste se za definiranje višestrukih tekstura, koje se potom miješaju i time se doprinosi vizualnom dojmu objekta zbog povećanog stupnja detalja. Navedeni UV slojevi pohranjuju se u obliku niza unutar svojstava *faceUvs i faceVertexUvs*. Moguće je postaviti i pripadnu zastavicu za ažuriranje - *Geometry.uvsNeedUpdate*[24].

Podatke iz transformacijske matrice moguće je izravno upisati u koordinate objekta, pomoću metode *applyMatrix(matrix).* Na ovaj se način optimira korištenje računalnih resursa. Prisutne su metode za izračun karakterističnih centroida svih prisutnih likova unutar objekta – *computeCentroids(),* normala likova i točaka – *computeFaceNormals() i computeVertexNormals*, tangenti – *computeTangents()* i graničnih okvira – *computeBoundingBox() i computeBoundingSphere()*[24].

Boje i koordinate moguće je definirati kao zasebne objekte (*eng. morph*) i označiti karakterističnim nazivom. Oni su tipa *morphColor* i *morphVertex*. Podržane su

zasebne metode za izračun nad takvim objektima, primjerice za izračun normale – *computeMorphNormals().* Na promjene na geometrijskim podacima moguće je reagirati korištenjem događaja.

## 5.3. Događaji

WebGL podržava osluškivanje događaja, odnosno karakteristične promjene na određenim podacima i djelovanje na iste. Klasa *EventDispacher* pomoću metoda *addEventListener(type, listener), removeEventListener(type, listener), hasEventListener(type, listener)*[22] upravlja upotrebom događaja. Prvi ulazni parametar – *type* označava vrstu događaja, a *listener* je poziv funkcije koja reagira na specifični događaj.

## 5.4. Geometrija spremnika

Podaci nužni za iscrtavanje pohranjuju se unutar pripadnog spremnika. Za pohranu podataka unutar spremnika koristi se *BufferGeometry* klasa[22]. Pohrana podataka unutar spremnika višestruko je korisna, prvenstveno zbog reduciranja procesorskih ciklusa i zauzimanja memorijskih resursa. Glavni je nedostatak složenost potrebnih izračuna unutar spremnika. Koristi se prvenstveno za statičke objekte. Unutar svojstava moguće je postaviti identifikacijsku oznaku instance klase, razmak (*eng. offset*), granični okvir (*eng. bounding box*), karakteristične atribute, te oznaku dinamike. Ukoliko je postavljen dinamički parametar (*eng. dynamic*) dolazi do dodatne pohrane određenih podataka unutar spremnika, sa svrhom bržih ažuriranja. U protivnom se zauzima manje memorije. Razmak je osobito važan kod indeksiranih spremnika, a predstavlja niz unutar kojeg su pohranjene pozicije na kojima dolazi do iscrtavanja sadržaja spremnika. Osim kvadratnog graničnog okvira (*eng. bounding box*) definiranog minimalnim i maksimalnim koordinatama vektora, moguće je koristiti i graničnu sferu[24] (*eng. bounding sphere*), kod koje je dovoljno definirati pripadni radijus. Korištenjem odgovarajućih metoda obavljaju se izračuni tangenti, normala, te graničnih okvira.

Također je moguće osluškivati događaje, odnosno promjene na podacima unutar spremnika.

## 5.5. Grafički objekti

Za potrebe grafičkih objekata unutar scene koristi se klasa *Object3D*. Objekti se identificiraju prema karakterističnoj identifikacijskoj oznaci i proizvoljnom imenu.

Lokalna pozicija objekta unutar scene definirana je u svojstvu *position*. Svojstvo *rotation* sadrži lokalnu orijentaciju objekta preko Eulerovih kutova, izraženu u radijanima. Podržano je proizvoljno postavljanje slijeda koordinatnih osi za Eulerove kutove. Alternativno, lokalna orijentacija objekta može se izraziti i u kvaternionima (*eng. quaternion*). U tom slučaju potrebno je postaviti zastavicu *useQuaternion*[24].

Lokalna transformacija objekta pohranjena je u njegovoj matrici (*eng. matrix*), a skaliranje unutar svojstva *scale*. Zastavica *frustumCulled* provjerava je li dio objekta vidljiv kameri, radi potreba iscrtavanja. Vidljivost cjelokupnog objekta moguće je postaviti pomoću zastavice *visible*. Zastavice za potrebe ažuriranja uključuju *matrixAutoUpdate* – za automatski utjecaj promjena na sadržaj matrice objekta, *matrixWorldNeedsUpdate* – za iniciranje potrebnih izračuna unutar matrice objekta, uslijed promjena podataka i *rotationAutoUpdate* – za izračun rotacijske matrice u svakom okviru prikaza (*eng. frame*). Specifično sjenčanje objekta postaje dijelom ukupne mape sjena pomoću svojstva – *castShadow*. Ukoliko je potrebno sjenčati materijal objekta koristi se *receiveShadow()*. Unutar objekta moguće je definirati i proizvoljne podatke u svojstvu *userData*.

Za svaki objekt moguće je definirati hijerarhijsku strukturu postavljanjem objekta roditelja (*eng. parent*) i djeteta (*eng. children*)[24]. Svaki objekt može imati jedan objekt roditelj i više objekata djece, koji se smještaju u niz *children*. Metoda *add(object)* dodaje *dijete - objekt* objektu za koji je pozvana. Za pretraživanje hijerarhije objekta koriste se metode *transverse(callback)* – *za* poziv funkcije

„callback“ za prvi objekt koji poziva ta funkcija i svu njegovu djecu, i *getDescendants()* – za dohvaćanje svih potomaka objekta.

Objekt je moguće proizvoljno usmjeriti pomoću funkcije *lookAt(vector).* Ulazni argument – vektor – predstavlja koordinatu točke prema kojoj se okreće objekt.

Kloniranje objekta obavlja se pozivom metode *clone().* Metoda *remove()* uklanja objekt iz scene. Matrice u kojima se nalaze pozicija, rotacija i skaliranje objekta moguće je jednostavno ažurirati djelovanjem metode *applyMatrix(matrix).*

Objekte je moguće pretraživati prema identifikacijskoj oznaci metodom *getObjectById(id, function)* ili prema imenu metodom *getObjectByName(name, function).* Prilikom translacije moguće je koristiti specijalne translacijske metode za pojedinu os – *translateX(distance), translateY(distance) i translateZ(distance)* ili univerzalnu translaciju metodom *translateOnAxis(axis, distance).*

Svaki objekt posjeduje vlastiti lokalni koordinatni sustav. Vektor u lokalnom sustavu ažurira se u skladu s pripadnim vektorom u globalnom sustavu pomoću metode *localToWorld(vector).* Obrnuti postupak omogućuje metoda worldToLocal(vector)[22].

## 5.6. Projektor

Klasa *Projector* upravlja projekcijom slike koju vidi kamera na platno HTML stranice. Osnova projekcije su projekcijski vektori, kojima nastaje prikaz na platnu, pomoću metode *projectVector(vector, camera)[24].* Prvi parametar predstavlja vektor, koji je potrebno projicirati. *Camera* predstavlja instancu klase *Camera*, koja se koristi unutar aplikacije. Navedena metoda prilikom projekcije modificira vektor. Obrnuti postupak izvodi metoda *unprojectVector(vector, camera).* Pri tome također dolazi do modifikacije vektora.

Za projekciju cjelokupne scene koristi se metoda *projectScene(scene, camera, sort).* Ulazni argumenti su instance klasa scene i kamere, te zastavica za korištenje Slikarskog algoritma. Slikarski algoritam[27] (*eng. Painter's algorithm*) predstavlja tehniku rješavanja problema vidljivosti i preklapanja objekata u

trodimenzionalnoj grafici. Naziv je dobio jer oponaša princip slikanja – slikanje započinje s najdaljim elementima i ide prema bližim. Na taj način, prvo se iscrtavaju najudaljeniji objekti. Iscrtavanjem bližih objekata dolazi do preklapanja dijelova već iscrtanih udaljenijih objekata i stvara se prizor koji odgovara prirodnim odnosima.

## 5.7. Ravnina projekcije

Prilikom iscrtavanja trodimenzionalne scene na platnu, potrebno je svakoj trodimenzionalnoj koordinati u stvarnom prostoru pridružiti pripadajuću dvodimenzionalnu koordinatu na platnu. Klasa za projiciranje scene na platno naziva se *Raycaster*[22]. Prilikom njezinog instanciranja potrebno je definirati vektor (trodimenzionalna koordinata, *eng. origin*), smjer zrake projekcije (*eng. direction*) i najbližu (*eng. near*) i najdalju ravnina (*eng. far*) koje omeđuju scenu. Dobivene vrijednosti ravnine projekcije nalaze se u intervalu između najbliže i najdalje ravnine.

Osim definicije zrake prema kojoj se odvija projekcija (*eng. ray*) i prethodno spomenutih ravnina, u svojstva se ubraja i parametar preciznosti. On predstavlja decimalnu vrijednost preciznosti preslikavanja zraka na ravninu projekcije.

Zraka projekcije postavlja se metodom *set(origin, direction*), koja kao ulazne parametre prima prvobitni vektor (*eng. origin*) i smjer[24]. Moguće je provjeravati presjek zrake s jednim objektom i njegovim potomcima u hijerarhiji – *intersectObject(object, recursive)* ili s više objekata i njihovim potomcima u hijerarhiji – *intersectObjects(object, recursive). Recursive* označava rekurzivno pozivanje metode za sve potomke objekta.

## 5.8. Osvjetljenje

Apstraktna klasa kojom je implementirano osvjetljenje u WebGL-u je *Light(hex),* a konstruktor kao ulazni parametar prima heksadecimalnu vrijednost

RGB komponenti boja. Unutar svojstva *color* postavlja se boja osvjetljenja. WebGL razlikuje ambijentalno, površinsko, usmjereno, hemisferalno, reflektorsko i točkasto osvjetljenje.

Ambijentalno[24] se osvjetljenje primjenjuje jednoliko na sve objekte u sceni :

```
var light = new THREE.AmbientLight( hex RGB );
```

Površinski izvor svjetlosti[24] (*eng. area light*) cijelom svojom površinom osvjetljava scenu. Osim heksadecimalne vrijednosti RGB parametara boje, potrebno je postaviti i intenzitet svjetla. Dimenzije izvora svjetlosti, njegovu poziciju i rotaciju moguće je proizvoljno postaviti.

```
var areaLight1 = new THREE.AreaLight( 0xffffff, 1 );

areaLight1.position.set( 0.0001, 10.0001, -18.5001 );

areaLight1.rotation.set( -0.74719, 0.0001, 0.0001 );

areaLight1.width = 10; areaLight1.height = 1;
```

Usmjereno osvjetljenje[24] (*eng. directional light*) utječe na objekte preko postavki materijala koje pozitivno i negativno utječu na njegov sjaj (*eng. Mesh_Lambert* i *eng*. *Mesh_Phong*):

```
var directionalLight = new THREE.DirectionalLight(hex, intensity, distance);
```

Konstruktor se od prethodnog osvjetljenja razlikuje u dodatnom parametru – udaljenosti do koje svjetlost dopire. Osvjetljenje će biti najveće blizu izvora, tj. na udaljenosti nula i postupno padati do definirane granične udaljenosti.

Isto kao površinsko osvjetljenje, na objekt djeluje i točkasti izvor svjetlosti (*eng. point light*). Razlika je u izgledu izvora svjetlosti. Naime, kod točkastog osvjetljenja izvor svjetlosti je točka u prostoru i nije podložan modifikaciji oblika, već isključivo promjeni pozicije.

```
var pointLight = new THREE.PointLight(color, intensity, distance);
```

Kod hemisferalnog osvjetljenja[24] (*eng. hemisphere light*) izvor svjetlosti se nalazi iznad scene.

```
var hemisphereLight = new THREE.HemisphereLight(skycolor, groundcolor, intensity,);
```

*Skycolor* predstavlja RGB parametre boje osvjetljenja koje djeluje na scenu, dok je u parametru *groundcolor* RGB vrijednost tla scene.

Reflektorsko osvjetljenje (*eng. spot light*) je osvjetljenje nastalo iz točkastog izvora svjetlosti s mogućnošću bacanja sjene u jednom smjeru.

```
var spotLight = new THREE.SpotLight( color, intensity, distance, castShadow );
```

Za reflektorsko osvjetljenje karakterističan je parametar *castShadow*, kojim se postavlja dinamičko korištenje sjena. Navedeni je postupak iznimno zahtjevan s obzirom na potrošnju računalnih resursa radi velikog broja potrebnih izračuna. Češće se koristi svojstvo *onlyShadow*, koji utječe isključivo na pozicioniranje sjene, bez utjecaja na osvjetljenje. Ovo se postiže postavljanjem parametra intenziteta na nulu, pri čemu se izbjegavaju dodatni izračuni. Za reflektorsko osvjetljenje moguće je definirati kut najvećeg dosega svjetlosti iz izvora, koji se zadaje u radijanima, a veći je od $\frac{\pi}{2}$ . Također, moguće je postaviti eksponencijalnu brzinu slabljenja svjetlosti u ovisnosti o udaljenosti od zrake (*eng. exponent*). Reflektorsko osvjetljenje jedino koristi mapu sjena (*eng. shadow map*). Sjenčanje utječe isključivo na područje koje je omeđeno vrijednostima iz svojstava *shadowCameraNear* i *shadowCameraFar*[22].

## 5.9. Strukture učitanih objekata

WebGL podržava učitavanje objekata različite vrste. Ovo je omogućeno unutar *Loader* klase za učitavanje. JSON format podataka općenito je prihvaćen za učitavanje u WebGL aplikacije.

Instanca klase *GeometryLoader* koristi se za učitavanje geometrije objekata iz JSON datoteke. Unutar metode *load(url)* kao ulazni parametar potrebno je navesti stazu do JSON datoteke, iz koje je potrebno učitati podatke.

U JSON formatu može biti definiran cjelokupni objekt. U tome se slučaju za učitavanje koristi instanca klase *JSONLoader()*[24]. Učitavanje obavlja metoda *load(url, callback, texturePath)*. Parametar *callback* odnosi se na funkciju koja se poziva prilikom učitavanja i u koju se smještaju učitani podaci. Obično je riječ o WebGL tipu podataka – *object*. *TexturePath* označava gdje je smještena tekstura objekta. Potrebno ga je navesti ukoliko se tražena tekstura ne nalazi u istoj mapi kao i JavaScript datoteka s programskim kodom aplikacije. U protivnom se izostavlja.

Osim objekta, u JSON datoteci može se pohraniti cjelokupna scena. Ona se kao takva učitava pomoću instance klase *SceneLoader* i metode *load(url, callback).* Parametar *callback* predstavlja instancu klase *Scene*, unutar koje se pohranjuje učitana scena. Instanca klase *TextureLoader*[22] koristi se prilikom učitavanje potrebnih tekstura objekata. Za razliku od ostalih učitavanja, kod učitavanja teksture nije moguće specificirati strukturu u koju će se spremiti učitani podaci, već je ona unaprijed određena. Učitavanje obavlja metoda *load(url).*

Učitavanje objekata prati se pomoću instance klase *LoadingMonitor*. Svaki učitani objekt potrebno je dodati pomoću metode *add(loader).* Klasa *LoadingMonitor* podržava metode za praćenje događaja.

### 5.10. Sat

Prilikom animacije potrebno je koristiti vrijeme. Za praćenje vremenskih intervala koristi se objekt klase *Clock*[24]. Unutar svojstava moguće je postaviti pokretanje sata nakon početnog ažuriranja podataka – *autoStart*. Vrijeme početka rada sata pohranjeno je u svojstvu *startTime*, a izražava se u broju proteklih milisekundi od 1. siječnja 1970. godine. Prethodno vrijeme potrebno radi ažuriranja pohranjuje se unutar svojstva *oldTime*. Vrijeme između dvaju mjerenja nalazi se unutar svojstva *elapsedTime*. Aktivnost sata provjerava se pomoću

svojstva *running*. Dostupne su metode za pokretanje sata - *start()* i njegovo zaustavljanje - *stop().* Proteklo vrijeme u mjernoj jedinici sekundi vraća metoda *getElapsedTime(),* a interval između dvaju poziva te metode vraća metoda *getDelta().*

# 6. Biblioteka Tween

Tween[28] predstavlja biblioteku za podršku grafičkim aplikacijama u programskom jeziku JavaScript. Inicijalno je Tween biblioteka razvijena za integraciju s bibliotekom EaseIJS, no omogućeno je i njeno samostalno korištenje. EaseIJS je biblioteka za HTML5 aplikacije.

Namjena Tween biblioteke je posredna integracija, tj. *tweening*. Naziv dolazi od *eng. inbetweening*, što označava proces generiranja posrednih (srednjih) okvira (*eng. frame*) između dvaju prikaza. Njegova je svrha kod korisnika stvoriti dojam kontinuiteta pokreta, odnosno da drugi od dvaju prikaza nastaje iz prvog. *Tweening* je općenito proces ključan za područje animacije , pa time i računalne animacije. U kontekstu digitalne animacije koristi se skraćeni naziv *tweening*. Napredne okoline za potporu grafičkoj animaciji omogućuju korisniku odabir specifičnih objekata unutar prikaza i definiranje njihove animacije, odnosno pokreta. Pritom sve međuradnje, odnosno transformacije i translacije objekta između početne i konačne točke animacije ostvaruje navedena okolina. Prijelazne je okvire moguće ručno podesiti ili se oni automatski generiraju pomoću interpolacije grafičkih parametara.

Ostvarivanje iluzije kontinuiranog pokreta osnovna je sastavnica svake grafičke aplikacije. Pojmovi *ease - in* i *ease - out* u digitalnoj animaciji označavaju mehanizam za definiranje fizike prijelaza između dvaju slijednih animacijskih stanja, tj. linearnost procesa *tweening*. Namjena biblioteke Tween jest upravo podrška navedenom mehanizmu.

Tween biblioteka omogućuje rad s svojstvima entiteta objekata, kao i s CSS svojstvima. Unatoč relativno jednostavnom aplikacijskom sučelju, Tween

biblioteku odlikuje jaka podrška implementiranju složenijih animacija pokreta pomoću intuitivnog ulančavanja naredbi.

Tween biblioteku karakterizira relativno dobra podržanost. Aplikacije koje je koriste stabilno se izvode u Internet preglednicima Google Chrome (verzija 22+), Safari (verzija 6+), Mozzila Firefox (verzija 15+), Internet Explorer (verzija 9+) i Opera (verzija 12+).

## 6.1. Postavke biblioteke Tween

Biblioteka Tween predstavlja podršku ostvarivanja kontinuirane animacije.

Korištenje Tween biblioteke potrebno je navesti unutar HTML definicije stranice pomoću naredbe:

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

Instanca klase Tween stvara se sljedećom naredbom :

```
var myTween = new Tween(object, property, easing, start, end, duration, suffixe);
```

Konstruktor klase Tween kao ulazne vrijednosti prima parametre objekt, svojstvo, *easing*, početna pozicija, krajnja pozicija, trajanje i sufiks[29].

Parametar objekt (*eng. object*) označava objekt u aplikaciji na koji je potrebno djelovati. Pojam objekt u ovom je slučaju općenit i odnosi se na bilo koji entitet koji je moguće svrstati u tip *Object* (*eng. type:Object*). Pritom objekt može biti vizualni dio prikaza, koji je potrebno animirati (npr. *document.body.style* – objekt kojim se definira izgled HTML stranice) ili anoniman objekt (*eng. new Object()*).

Svojsto (*eng. property*) je parametar znakovnog tipa (*eng. type:String*). Unutar deklaracije navodi se naziv parametra prethodno navedenog objekta, koji *Tween* metoda ažurira. Unutar stvaranja instance klase *Tween*, moguće ga je izostaviti.

Pojmom *easining* označava se funkcija koja se primjenjuje na animaciju kretanja objekta (*eng. type:Function*).

Parametri početna pozicija (*eng. start*) i krajnja pozicija (*eng. end*) predstavljaju brojčane vrijednosti (*eng. type:Number*) koordinata inicijalne i konačne pozicije objekta nakon animacije. Brojčane je vrijednosti i parametar trajanje (*eng. duration*), kojim se definira trajanje animacije u sekundama.

Posljednji parametar – *sufiks* (*eng. suffixe*) znakovnog je tipa (*eng. type:String*), a njime se označava jedinica, koja se pridjeljuje vrijednosti svojstva na koje se djeluje. Primjerice sufiks „*pt*“ označava točku (*eng. point*), dok sufiks „*em*“ označava mjernu jedinicu iz područja tipografije, koja je jednaka specificiranoj veličini točke (*eng. point size*). Sufiks može biti i specijalan znak, primjerice „*%“* – kojim se označava postotak.

## 6.2. Metode

Metoda *Tween.start()* započinje izvođenje *tween* animacije od definirane početne točke. Ona ne posjeduje ulazne parametre i nema povratnih vrijednosti. Često se koristi za ponavljanje animacije ispočetka, u slučaju neočekivanog zastoja ili nakon uspješnog završetka[29].

*Tween.rewind()* je metoda za vraćanje animacije na početak, tj. povratak u stanje koje prethodi izvođenju animacije. Pri tome se svi ažurirani parametri vraćaju na početne vrijednosti. Ukoliko se metoda *Tween.rewind()* poziva tijekom izvođenja animacije, dolazi do povratka u prethodno stanje (neposredno prije početka animacije) i animacija se nastavlja. Ako je metoda pozvana nakon zaustavljanja ili završetka animacije, također dolazi do povratka u prethodno stanje, ali animacija ostaje neaktivna. Preporučuje se korištenje metode *Tween.rewind()* nakon što je animacija zaustavljena metodom *Tween.stop()*. Nastavak animacije nakon njenog zaustavljanja naredbom *Tween.stop()*, obavlja naredba *Tween.resume()*. Za razliku od *Tween.rewind()*, kod metode *Tween.resume()* animacija se nastavlja od trenutnog stanja, bez povrataka u početno stanje[29].

Metoda *Tween.yoyo()* pokreće animaciju unatrag, dok metoda *Tween.fforward()* uzrokuje trenutnu transformaciju u završno stanje nakon animacije.

Tween metodu moguće je pozvati na određeni događaj. Metoda *Tween.addListener(listenerObject)* povezuje promjene vezane uz *listenerObject* s izvođenjem određene metode. Objekt na čije promjene reagira slušatelj događaja (*eng. event listener*) može se i ukloniti pomoću metode *Tween.removeListener(listenerObject).*

Metoda *Tween.continueTo(end, duration)* uzrokuje nastavak animacije objekta od trenutne pozicije do nove konačne vrijednosti, definirane parametrom *end*. Postavlja se i nova vrijednost trajanja animacije[29].

### 6.3. Funkcije *Ease* i događaji

Funkcije *Ease* opisuju međukorake animacije od početne do konačne pozicije. Tween biblioteka podržava petnaest načina animacije pokreta. Obična animacija definirana je s prefiksom *regular*. Objekt je moguće animirati s poskakivanjem, tj. *bounce*. Oštar prijelaz iz početnog u konačno stanje definiran je funkcijom *strong*. Animacija koja odstupa od početnih i krajnjih vrijednosti ostvaruje funkcijom *back*, a efekt elastičnosti dobiva se pomoću funkcije s prefiksom *elastic*[29].

Funkcija brzine svake od navedenih funkcija ovisi o karakterističnom sufiksu. *EaseIn* označava početnu brzinu jednaku nuli, nakon čega dolazi do akceleracije, a najveća se brzina postiže u posljednjem koraku animacije. *EaseOut* djeluje obrnutim postupkom. Početna vrijednost funkcije brzine jednaka je najvećoj brzini, nakon čega se odvija deceleracija. Konačna vrijednost funkcije brzine jednaka je nuli. Moguća je i kombinacija navedenog. *EaseInOut* započinje s brzinom jednakom nuli. Prvu polovicu animacije obilježava akceleracija do određene vrijednosti brzine. Potom nastupa deceleracija, kojom na kraju animacije brzina poprima vrijednost nula.

Moguće su funkcije:

- *Tween.regularEaseIn*
- *Tween.regularEaseOut*
- *Tween.regularEaseInOut*
- *Tween.bounceEaseIn*
- *Tween.bounceEaseOut*
- *Tween.bounceEaseInOut*
- *Tween.strongEaseIn*
- *Tween.strongEaseOut*
- *Tween.strongEaseInOut*
- *Tween.backEaseIn*
- *Tween.backEaseOut*
- *Tween.backEaseInOut*
- *Tween.elasticEaseIn*
- *Tween.elasticEaseInOut*

Biblioteka Tween podržava šest vrsta događaja. S obzirom da se događaji odnose na karakterističnu animaciju objekta koji se osluškuje, moguće ih je podijeliti na početak kretanja (*eng. onMotionStarted(eventObject)*), završetak kretanja (*eng. onMotionEnded (eventObject)*), nastavak kretanja (*eng. onMotionResumed(eventObject)*), zaustavljanje kretanja (*eng. onMotionStopped (eventObject)*), ponavljajuće kretanje (*eng*. *onMotionLooped (eventObject)*) i promjenu kretanja (*eng. onMotionChanged(eventObject)*)[29].

Objekt na kojem se odvija događaj posjeduje dva svojstva – instancu objekta i tip događaja. Instanca objekta je objekt tipa *Tween* (*eng. type:Tween()*), dok je tip događaja naziv jedne od navedenih vrsta događaja (*eng. type:String*).

Događaj je moguće deklarirati povezivanjem metode instance objekta *Tween* istog naziva kao i određeni događaj:

```
tweenEvent_1=new Tween (document.getElementById('sqA').style, 'left',
Tween.elasticEaseOut,0,500,1,'px');
tweenEvent_1.onMotionStarted = function(){alert( 'Motion has started' )};
tweenEvent_1.start();
```

Alternativno korištenje događaja uključuje korištenje *Tween.addListener()* metode:

```
tweenEvent_2=new Tween( document.getElementById('sqA').style, 'left',
Tween.elasticEaseOut,0,500,1,'px');
var a = new Object();
a.onMotionStarted=function(){ alert('Motion has started')};
tweenEvent_2.addListener(a)
tweenEvent_2.start();
```

Prednost drugog načina jest mogućnost korištenja proizvoljnog broja slušatelja događaja, te njihovo jednostavno uklanjanje pomoću naredbe *Tween.removeListener().*

## 6.4. Dodatne biblioteke

Metode i događaje biblioteke Tween moguće je, osim za objekte, koristiti i za tekst, boju i prozirnost[29]. Dostupne su zasebne Tween biblioteke za svaki od navedenih vrsta objekata.

Klasa *ColorTween* je svojevrsna podklasa klase *Tween*. Stoga je unutar projekta unutar kojeg se koristi, nužno deklarirati i korištenje *Tween* klase:

```
[script language="javascript" src="Tween.js"][/script]

[script language="javascript" src="ColorTween.js"][/script]
```

Instanca klase *ColorTween* deklarira se naredbom:

```
var colorTween = new ColorTween( object, property, easing, startColor,
endColor, duration);
```

Parametrima *startColor* i *endColor* postavljaju se početna i konačna vrijednost boje objekta na koji se djeluje. Nakon izvođenja *Tween* metode *Tween.start()* dolazi do kontinuirane animacije prijelaza boje objekta iz početne u konačnu.

Klasa *OpacityTween* također predstavlja proširenje *Tween* klase. Namjena joj je upravljanje parametrom prozirnosti vizualnih objekata.

Uz definiciju korištenja *OpacityTween* klase, potrebno je istaknuti i uporabu *Tween* klase:

```
[script language="javascript" src="Tween.js"][/script]
[script language="javascript" src="OpacityTween.js"][/script]
```

Instanca klase OpacityTween deklarira se naredbom:

```
var opacityTween = new OpacityTween(Object, easing, startOpacity,
endOpacity, duration);
```

Karakteristični parametri klase *OpacityTween* su *startOpacity* i *endOpacity*, kojima se definiraju početna i konačna vrijednost prozirnosti objekta. Kontinuirana animacija prijelaza između dvaju prozirnosti odvija se nakon pokretanja naredbe *start().*

Klasa *TextTween* upravlja efektima animacije tekstualnih objekata. Zajedno sa klasom *TextTween* deklarira se i korištenje klase *Tween*:

```
[script language="javascript" src="Tween.js"][/script]
[script language="javascript" src="TextTween.js"][/script]
```

Instanca klase *TextTween* deklarira se naredbom:

```
var t = new TextTween(object,property,text,easing,duration);
```

Naredbom se djeluje na tekstualno svojsto navedeno unutar parametra *property*.

# 7. Pojednostavljena Simple Tween biblioteka

Biblioteka Simple Tween[30] predstavlja mikrookvir (*eng. microframework*) za razvoj grafičkih aplikacija, koji je relativno jednostavno integrirati smještanjem datoteke koja sadržava Simple Tween biblioteku unutar projekta. Njezino je izvođenje izrazito optimirano s obzirom na performase i pripadnu potrošnju resursa. Originalno je osmišljena za razvoj aplikacija koje koriste HTML5-ov element platno, no moguće ju je koristiti za sve aplikacije implementirane u programskom jeziku JavaScript. Predstavlja pojednostavljenje Tween biblioteke.

## 7.1. Postavke i metode biblioteke

Korištenje Simple Tween biblioteke potrebno je definirati unutar HTML stranice pomoću naredbe:

```
<script type="text/javascript" src="tween.min.js"></script>
```

Za deklaraciju instance klase *Tween* potrebni parametri uključuju početnu vrijednost (*eng. start value*), udaljenost (*eng. distance*), trajanje (*eng. duration*), vrstu animacije (*eng. animation type*) i petlju unutar koje se odvija animacija[30] (*eng. loop*).

```
var myTween = new Tween(startValue, distance, duration, animationType, loop);
```

Početna vrijednost predstavlja parametar od kojeg počinje proces *tweeninga*, tj. kontinuiranog animiranog prijelaza do ciljne vrijednosti, koja se nalazi na definiranoj udaljenosti istoimenog parametra proizvoljne jedinice. Trajanje animacije izražava se u milisekundama. Animacija se izvodi slijedom naredbi koje se nalaze unutar petlje.

Vrijednosti parametara instance klase *Tween* moguće je naknadno izmijeniti pomoću naredbe:

```
myTween.set(startValue, distance, duration, animationType, loop);
```

Trenutnu vrijednost parametra na koji se djeluje moguće je provjeriti u proizvoljnom trenutku pomoću naredbe:

```
myTween.getValue();
```

Isti je parametar moguće postaviti na početnu vrijednost naredbom :

```
myTween.reset();
```

Provjera aktivnosti animacije vraća *boolean* vrijednost i obavlja se metodom:

```
myTween.expired();
```

## 7.2. Vrste animacije

Simple Tween biblioteka podržava 22 vrste animacija. Moguće ih je svrstati unutar osam osnovnih kategorija.

*Linear* označava jednoličnо pravocrtno gibanje. Brzina je pri tome jednolika, bez akceleracije ili deceleracije. Ostale kategorije omogućuje odabir *easeIn*, *easeOut* ili *easeInOut* opcije, istovjetne opisima unutar poglavlja Biblioteka Tween. *Quad* odlikuje kvadratna funkcija brzine. Kubična funkcija brzine obilježje je kategorije *Cube*. Red polinoma funkcija *Quart* i *Quint* odgovaraju odgovarajućim skraćenicama. Za *Quart* on iznosi četiri, a za *Quint* pet. Sinusna funkcija brzine karakteristična je za kategoiju *Sine*. Eksponencijalna funkcija brzine odlikuje kategoriju *Expo*, dok je kružna funkcija brzine obilježje posljednje kategorije – *Circ*[30].

## 7.3. Proširenje funkcionalnosti

Simple Tween biblioteka je implementirana sa svojstvom prototipnog nasljeđivanja. Stoga je moguće njezino intuitivno proširivanje primjenom principa objektno orijentirane paradigme.

Primjerice moguće je implementirati mogućnost promjene trenutnog načina animacije.

```
Tween.prototype.setAnimation = function (vlastita_Animacija) {
this.animationType = vlastita_Animacija;};
```

U navedenom primjeru korisnički definirana animacija, sadržana unutar funkcije *vlastita_Animacija*, zamjenjuje animaciju koja se trenutno koristi.

Simple Tween biblioteka općenito koristi funkciju *Date.now()* za dohvat vremenskih vrijednosti potrebnih za parametar trajanja. Izvođenje je moguće ubrzati ukoliko se koristi prethodno pohranjena vrijednost vremena. Tada je potrebno sa istom zamijeniti reference na funkciju *Date.now()*.

## 7.4. Sekvenca naredbi

Složenije animacije moguće je ostvariti ulančanim pozivima određenih metoda. Sekvencu poziva moguće je ostvariti sa ili bez *tweening-a*.

Primjerice, ukoliko je potrebno promijeniti *x*-koordinatu objekta na vrijednost 400, unutar animacije trajanja 500 milisekundi, a potom nakon 500 milisekundi kontinuirano izmijeniti (*eng. tween*) vrijednost alfa parametra na vrijednost 0 unutar 2 sekunde, postaviti joj vidljivost na vrijednost *false* i pozvati *onComplete()* funkciju, pripadni slijed naredbi je :

```
var myTween =  Tween.get(myTarget).to({x:400},500)
.wait(500).to({alpha:0},2000).set({visible:false}).call(onComplete);
```

Sekvenca naredbi ne treba sadržavati *tweening*. Općenita sintaksa takve sekvence je :

```
var mySeq = Tween.get(target).call(doStuff,[param])
.wait(500).set({prop:value}).set({prop:value},foo).call(allDone);
```

U navedenom primjeru se nakon 500 milisekundi postavlja proizvoljni parametar funkcije *foo*, nakon čega se poziva funkcija *allDone*.

# 8. Eksplozijski dijagram

Računalna grafika nastoji predočiti određene pojave, predmete i događaje na način koji je prihvatljiv promatraču, bilo u realnom vremenu ili stupnju detalja prikaza. U suvremenoj grafici težište se stavlja na trodimenzionalne objekte, kao što su mehanički sklopovi, električni uređaji, arhitektonski prikazi, te živa bića. Navedene objekte nerijetko karakterizira složena unutarnja struktura.

Eksplozijski dijagrami[31] predstavljaju način predočavanja složene unutarnje strukture objekta. Na ovaj se način međusobno odvajaju dijelovi, kako bi se istaknuli unutarnji dijelovi, sakriveni vanjskim slojem. Cilj eksplozijskih dijagrama nije samo prikazati unutarnje dijelove, nego i globalnu strukturu objekta, koja uključuje lokalne prostorne odnose između dijelova. Time se daje detaljan pregled svakog pojedinog dijela, kao i prostorna geometrija objekta.

Eksplozijske dijagrame moguće je podijeliti na statične i dinamične. Statični su dijagrami u nepovoljnijoj poziciji, prvenstveno zbog ograničenog pretraživanja objekta. Naime, često su neprikladni za usredotočeno pretraživanje pojedinog dijela složene strukture objekta, i pripadajućih poddijelova. Obično eksplodira cijeli objekt, tj. dolazi do odvajanja svih dijelova, što zbog većeg broja dijelova uzrokuje nepreglednost prikaza. U takvim okolnostima, korisnicima je otežano snalaženje u prikazu, te je potrebno više vremena kako bi se pronašao traženi dio objekta. Također, ukoliko je riječ o običnoj eksploziji, odnosno odvajanju, dolazi do nevjerodostojnog razmještaja. Time dijelovi objekta između kojih postoji prostorna povezanost, istu gube, što dodatno otežava snalaženje na eksplodiranom prikazu. Poziciju i orijentaciju dijelova koji su u centru fokusa promatrača ovim načinom nije moguće utvrditi, što uvelike narušava preglednost. Običnim statičnim dijagramom nije moguće postići odgovarajuću razinu detalja, čime se narušavaju temelji suvremene računalne grafike.

Prilikom implementacije eksplozijskog dijagrama preporučuje se dinamični pristup. Priroda dinamičnog pristupa isključuje najveći dio ograničenja, koja su narušavala izvedbu statičnog eksplozijskog dijagrama. Korisnicima se omogućuje uvid u prostorne odnose između pojedinih dijelova trodimenzijskog objekta, kao i olakšano snalaženje i pronalaženje specifičnih dijelova. Prilikom implantacije

potrebno je pripaziti na odgovarajuće smjerove odvajanja pojedinih dijelova. Osim zbog očuvanja preglednosti prostornih veza, nužno je osigurati vidljivost pojedinih dijelova nakon eksplozije, odnosno ograničiti preklapanje.

Također, korisniku se omogućuje upravljanje eksplozijskim dijagramom. Poželjno je koristiti stupnjevanje eksplozijskog dijagrama, tj. odvajanje podijeliti na manje cjeline, u svrhu bolje preglednosti. Početni stupnjevi označavaju odvajanje vanjskih dijelova. Porastom numeričke vrijednosti oznake pojedinog stupnja, dolazi do odvajanje dijelova sve bliže centru objekta, odnosno dubljeg sloja. Korisniku je potrebno omogućiti efikasno upravljanje eksplozijskim dijagramom. Najinuitivniji način upravljanja jest pomoću klika mišem na objekt. Pozicijom klika moguće je odabrati odgovarajući dio objekta, koji potom eksplodira. Lociranje klika na prikazu moguće je implementirati podržanim funkcijama, ovisno o tehnologiji izvedbe. Alternativa lociranju klika mišem su gumbi za upravljanje stupnjem eksplozije. Ovim se načinom olakšava implementacija, radi izostanka lociranja. Također, doprinosi preglednosti zbog slojevitog načina eksplozije dijelova, od vanjskih prema unutarnjim.

## 8.1. Postavke eksplozijskih dijagrama

Prilikom implementacije eksplozijskih dijagrama potrebno je definirati smjerove eksplozije dijelova trodimenzijskog objekta, kao i razmak između susjednih dijelova[32].

### 8.1.1. Vidljivost i kanonski smjerovi eksplozije

Vidljivost predstavlja osnovnu postavku eksplozijskih dijagrama. Potrebno je odabrati razmak koji osigurava vidljivost svih eksplodiranih dijelova. Ovom se postavkom osigurava svojstvo preglednosti.

Kanonski smjerovi eksplozije koriste se zbog sličnosti sa stvarnom orijentacijom objekata. Naime, većinu trodimenzijskih objekata odlikuje tzv. okvir kanonskih

koordinata (*eng. canonical coordinate frame*). Definicija okvira kanonskih koordinata zadana je faktorima kao što su simetrija, orijentacija stvarnog svijeta i domenski specifične konvencije[32]. Ograničavanje smjerova eksplozije doprinosi olakšanju interpretacije korisnika na koji je način eksplozija utjecala na pomak pojedinog dijela objekta od njegove početne pozicije. Također, olakšava se implementacija i smanjuje potrošnja računalnih resursa potrebnih za izvođenje aplikacije. Uobičajeno je ograničiti eksploziju na šest smjerova, na način da je prisutna okomica između susjednih smjerova. Kanonski smjerovi eksplozije također doprinose preglednosti.

### 8.1.2. Preklapanje i kompaktnost

Prilikom definicije smjerova eksplozije dijelova objekta potrebno je spriječiti preklapanje pojedinih dijelova. Odvajanje dijelova se obavlja na način da se smjerovi odvajanja ne sijeku (osim zajedničkog ishodišta). Stoga je korisno slijediti prethodno navedeno pravilo o okomici između susjednih smjerova. Rezultirajući raspored dijelova na konačnom prikazu omogućuje predočavanje prostornih povezanosti i preklapanja, te relativnih pozicija dijelova.

Kompaktnost je poželjno svojstvo prikaza uslijed djelovanja eksplozijskog dijagrama. Na kompaktnost ponajviše utječe odabir odgovarajućeg razmaka između dijelova. Veličinu razmaka poželjno je minimizirati. Naime, minimiziranje relativne udaljenosti određenog dijela od njegove početne pozicije doprinosi efikasnijem prikazu, tj. korisniku je olakšana interpretacija rekonstrukcije objekta. Također, ovom se postavkom doprinosi racionalizaciji iskorištenog prostora, odvojeni dijelovi zauzimaju manji prostor na ekranu.

### 8.1.3. Hijerarhija dijelova

Hijerarhija dijelova je važna postavka eksplozijskog dijagrama, koja prvenstveno definira prethodno spomenuto stupnjevanje eksplozije. Složeni objekti

posjeduju hijerarhijsku strukturu dijelova, kojom su definirani odnosi roditelj – dijete, odnosno glavni dio i njegovi dijelovi. Potrebno je nakon odvajanja zadržati vizualnu percepciju hijerarhije dijelova. Prilikom stupnjevanja eksplozije prvo se međusobno odvajaju glavni dijelovi, a potom njihovi dijelovi u narednim stupnjevima eksplozije. Složenost hijerarhije dijelova ovisi o stupnju detalja dijelova koje je potrebno prikazati. Objekte smanjene razine detalja odlikuje podjela na jedinstvene dijelove, koje u narednim odvajanjima nije potrebno rastavljati na poddijelove.

## 8.2. Postavke rastavljanja dijelova

U suvremenoj računalnog grafici prisutni su brojni pristupi rastavljanja složenih objekata na pripadajuće dijelove. Najčešće se koristi kontekstualno rastavljanje i spremnik rastavljanja.

### 8.2.1. Kontekstualno rastavljanje

Odabirom kontekstualnog rastavljanja unosi se poboljšanje u vidu podizanja razine konteksta eksplozijskog dijagrama. Ovim načinom korisnik dodatno stječe uvid u pojedine sastavnice konstrukcije objekta. Kod kontekstualnog rastavljanja dolazi do početnog grupiranja izvedbeno bliskih sastavnica, primjerice dijelova koji povezani obavljaju određenu funkciju cjelokupnog objekta. Korisnik time dobiva strukturirani početni pregled i mogućnost odabrati kontekstualnu skupinu dijelova koje će dodatno proučavati. Važno je napomenuti da dijelovi objekta koji pripadaju pojedinoj kontekstnoj skupini nisu nužno i prostorno bliski unutar konstrukcije objekta.

### 8.2.2. Spremnik rastavljanja

Rastavljanje dijelova s obzirom na tzv. spremnik rastavljanja najčešće je zastupljeno kod objekata s razvijenom hijerarhijskom strukturom. Glavna odrednica njihove hijerarhije jest sadržanost poddijelova (*eng. child part*) unutar glavnih dijelova (*eng. parent part*)[32]. Pritom glavni dijelovi predstavljaju granice razdvajanja ostalih dijelova cjelokupnog objekta, odnosno spremnike rastavljanja.

U svrhu očuvanja vizualnosti relacija unutar pojedinog spremnika rastavljanja, on se rastavlja pripadnom ravninom odsijecanja. Navedena ravnina odsijecanja prolazi središtem spremnika i dijeli ga na dva dijela. Normala ravnine odsijecanja paralelna je odabranom smjeru eksplozije. Potom dolazi do razdvajanja dvaju novonastalih dijelova spremnika, kako bi se otkrili sadržani dijelovi. Postupak se ponavlja uzastopno do otkrivanja svih sadržanih dijelova. Pritom svakom iteracijom postupka novonastali dio spremnika postaje spremnik razdvajanja i ponavlja se postupak odsijecanja. Kako bi se naglasila pripadnost pojedinih dijelova istom početnom spremniku, odabire se ravnina odsijecanja odgovarajuće orijentacije u svrhu smanjivanja udaljenosti na koju je potrebno razdvojiti segmente spremnika rastavljanja, kako bi se otkrili sadržani dijelovi u njemu.

Rastavljanje pomoću spremnika rastavljanja zasniva se na održavanju prostorne bliskosti, odnosno dijelovi koji se nalaze u blizini u početnom objektu, zadržat će privid relacije bliskosti i nakon razdvajanja.

## 8.3. Slijed eksplozije objekta

U svrhu omogućavanja interaktivnih eksplozijskih dijagrama potrebno je definirati karakteristični graf eksplozije. Graf eksplozije predstavlja strukturirani slijed eksplozije objekta, odnosno razdvajanja pojedinih dijelova. Osnova grafa jest hijerarhija objekta, tj. relacije između pojedinih dijelova. Potrebno je osigurati da eksplozija objekta ne naruši definirana ograničenja. Stoga je unutar slijeda eksplozije nužno spriječiti da pojedini dio eksplodira prije nego što su to učinili

njegovi prethodnici. Unutar strukture objekta dijelovi su prostorno raspoređeni bliže središtu ili bliže vanjskim rubovima objekta. Prethodnici pojedinog dijela su dijelovi koji se u odnosu na navedeni dio nalaze bliže vanjskom rubu objekta. Oni nužno trebaju eksplodirati prije, kako bi se spriječilo preklapanje, koje narušava preglednost. Također, potrebno je optimirati razdvajanje.

Općenito za svaki dio *d* objekta *o* (*d ϵ o*) potrebno je definirati smjer eksplozije i udaljenost na koju se taj dio pomiče nakon eksplozije. Smjer i udaljenost dijela $d_1$ ovisi o smjeru i udaljenosti dijela $d_2$, ako je $d_2$ prethodnik od $d_1$. Na ovaj način se posredno potiče pomicanje manjih dijelova zajedno s većim, čime se doprinosi preglednosti.

Dijelovi $d_1$ i $d_2$ objekta *o*, s obzirom na prostorne relacije, mogu se međusobno doticati, preklapati i sadržavati. Priroda njihovih međusobnih relacija uvelike utječe na definiranje smjera i udaljenosti eksplozije. Dio $d_1$ će sadržavati dio $d_2$, ukoliko se točka koja pripada rubu dijela $d_2$ i najudaljenija je od središta dijela $d_2$ nalazi unutar dijela $d_1$. Ukoliko se u ovom slučaju odabere kontekstno razdvajanje, svaki od navedenih dijelova grupirat će se s kontekstno bliskim dijelovima i u ovisnosti o njima dalje razdvajati. U slučaju principa spremnika razdvajanja, dio $d_1$ će se podijeliti na dva segmenta, koja će se potom razdvojiti kako bi se otkrio dio $d_2$.

Relacija međusobnog doticanja može uključivati i blokiranje, na način da se niti jedan od deblokiranih dijelova ne može pomaknuti. U navedenom slučaju riječ je o međusobno isprepletenim dijelovima. Blokiranje u ovom slučaju nije uzrokovano nekim drugim dijelovima objekta, već isključivo isprepletenim dijelom. Pristup u navedenim okolnostima uključuje definiranje najvećeg parcijalnog podskupa dijelova, koji se početno razdvaja, deblokirajući time ostatak dijelova objekta koji se tada mogu neovisno pomicati. U slučaju višestruke isprepletenosti postupak se iterativno ponavlja, do oslobađanja svih uključenih dijelova.

Pojava međusobne isprepletenosti može uključivati glavni dio, kojeg blokiraju u njemu sadržani dijelovi. U ovom slučaju riječ je o pristupu spremnika rastavljanja, te se glavni dio, odnosno spremnik, dijeli na dva individualna segmenta. Smjer eksplozije određen je smjerom pogleda korisnika. Novonastale segmente potrebno je razdvojiti na udaljenost, koja omogućuje daljnje odvajanje sadržanih dijelova. S

obzirom na postavke eksplozijskih dijagrama, navedenu je udaljenost potrebno optimirati, odnosno svesti na najmanju moguću.

### 8.3.1. Prostorno utemeljen slijed eksplozije

Slijed eksplozije dijelova je općenito definiran prema prostornom rasporedu. Definiranje slijeda eksplozije ovim pristupom predstavlja iterativan postupak koji se zasniva na neovisnom pomicanju. Koncept neovisnog pomicanja predstavlja mogućnost pomicanja u nekom od kanonskih smjerova, pri čemu nije potrebno prethodno pomaknuti neki drugi dio objekta. Dijelovi s inicijalnom mogućnošću neovisnog pomicanja ulaze u prvu razinu slijeda eksplozije. Nakon njihovog pomicanja u određenom kanonskom smjeru, postupak se ponavlja za dijelove koji su novonastalom situacijom dobili mogućnost neovisnog pomicanja. Oni se smještaju u naredne razine slijeda eksplozije. Postupak se iterativno ponavlja sve dok se u slijedu eksplozije ne nalaze svi dijelovi objekta.

### 8.3.2. Hijerarhijski utemeljen slijed eksplozije

Ukoliko je prisutna hijerarhijska struktura objekta, ona postaje osnova za definiranje slijeda eksplozije. Hijerarhijska je struktura uobičajeno definirana pomoću odgovarajućeg strukturiranog grafa hijerarhije. Unutar grafa su jasno definirani odnosi između dijelova objekta. Ovim se pristupom omogućava prostorno neovisno eksplodiranje dijelova, isključivo ovisno o grafu hijerarhije. Pri tome eksplozija skupine dijelova direktno utječe na njihove sljedbenike u grafu hijerarhije, tj. na konačan pomak dijelova koji su njihovi sljedbenici utjecat će njihovi pomaci uslijed eksplozije.

# 9. Implementacija

Za potrebe rada implementirana je Web aplikacija, koja demonstrira strukturu uređaja Microsoot. Aplikacija koristi već generirani model uređaja Microsoot, koji se sastoji od 71 zasebnog dijela. Model je napravljen u razvojnoj okolini Blender. Unutar aplikacije koristi se u JSON formatu, na način da se zasebno učitava svaki od dijelova. Prilikom prevođenja u JSON format korištena je preuzeta skripta[33].

Aplikacija je napisana u standardu HTML5 i koristi skriptu *myScript*, koja je implementirana u programskom jeziku JavaScript. Kao potpora implementaciji korištene su dodatne biblioteke Three[23] i SimpleTween[28]. Svi se elementi aplikacije dodaju unutar spremnika[34], koji implementiran pomoću elementa *Container*.

Prilikom definicije postavki scene korištene su instance predefiniranih klasa biblioteke Three. Aplikacija koristi perspekcijsku kameru naziva *mainCamera*:

```
camera = new THREE.PerspectiveCamera(fov, width/height, 0.1, 100);

camera.name = "Main Camera";
```

Pozicija kamere zadana je pomoću trodimenzionalnog vektora. Općenito se unutar aplikacije za sve pozicije koristi trodimenzionalni vektor, iz biblioteke Three:

```
var vector = new THREE.Vector3();
```

Scena je također instanca predefinirane klase biblioteke Three[23]:

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

scene.name = "The Scene";
```

Iscrtavanje se obavlja pomoću instance klase *Three.WebGLRenderer.*

```
renderer = new THREE.WebGLRenderer();

renderer.name = "Renderer";

renderer.setSize(width, height);
```

Interakcija aplikacije s mišem implementirana je pomoću *Three* objekta *TrackballControl.* Pomoću miša moguće je rotirati objekt, približavati (*eng. zoom*) i udaljavati kameru, odvajati pojedine dijelove i upravljati eksplozijom objekta.

```
controls = new THREE.TrackballControls( camera, renderer.domElement );

controls.name = "Trackball Controller";
```

Aplikacija koristi ambijentalno i usmjereno osvjetljenje. Prisutno je prednje i stražnje usmjereno osvjetljenje. Osvjetljenje je implementirano pripadnim *Three* objektima *Three.AmbientLight* i *Three.DirectionalLight*.

Projektor i ravnina projekcije također su *Three* objekti:

```
projector = new THREE.Projector();

plane = new THREE.Mesh( new THREE.PlaneGeometry( 2000, 2000, 8, 8 ), new
THREE.MeshBasicMaterial( { color: 0xffffff, opacity: 0.25, transparent:
true, wireframe: true } ) );
```

Interakcija s mišem i pripadno iscrtavanje implementirano je pomoću događaja. Razlikuje se klik (*eng. mouse down*), otpuštanje (*eng. mouse up*) i pomicanje (*eng. mouse move*) miša. Uslijed ovih događaja poziva se pripadna funkcija, nakon koje dolazi do ponovnog iscrtavanja prikaza, kako bi se obuhvatile promjene. Iscrtavanje se obavlja unutar funkcije *renderer().* Ažuriranje parametara interakcije s mišem i poziv iscrtavanja obavlja se unutar funkcije *animate().*

Detekcija odabira određenog dijela provjerava se pomoću presjeka zrake projekcije i modela. Zraka projekcije (*eng. raycaster*) definirana je pomoću točke i smjera. Točka koja definira zraku projekcije jednaka je trenutnoj poziciji kamere percepcije. Smjer zrake projekcije definiran je pomoću vektora, čija je početna točka pozicija kamere percepcije, a konačna pozicijom miša na platnu. Zraka projekcije presijeca model cijelom njegovom dubinom. Dijelovi koje sječe dobiveni su pomoću metode *intersectObject* i smještaju se u varijablu *intersect*, a prvi od njih je odabrani dio.

```
var raycaster = new THREE.Raycaster( camera.position, vector.sub(
camera.position ).normalize() );

var intersects = raycaster.intersectObjects( objects );
```

```
if ( intersects.length > 0 ) {

controls.enabled = false;

SELECTED = intersects[ 0 ].object;
```

Klikom miša upravlja se eksplozijom modela. Eksplozija modela omogućena je jedino unutar opcije rastavljanja. Pomoću kontrolirane eksplozije dolazi do rastavljanja modela na dijelove i njihovog razdvajanja. Eksplozija se odvija u stupnjevima. Model se izlaže prema principu „*izvana prema unutra*“, tj. rastavlja se počevši od rubnih vanjskih dijelova.

Model karakteriziraju samostalni dijelovi bez pojave isprepletenih dijelova. Spremnici unutar kojih se nalaze unutarnji dijelovi nisu kompaktni, tj. sastoje se od više dijelova. Stoga nije potrebno provoditi rastavljanje pomoću spremnika. Hijerarhija dijelova nije početno definirana, već su dijelovi podijeljeni u skupine prema prostornoj poziciji. Navedene skupine odgovaraju stupnjevima eksplozije. Prisutno je pet skupina u kojima su dijelovi podijeljeni prema položaju u modelu.

Eksplozijom se upravlja pomoću klika mišem na proizvoljni dio. Na odabrani dio prenosi se fokus kamere perspekcije i odvija se eksplozija skupine dijelova, kojima pripada navedeni element. Odabirom sve dublje smještenih elemenata dolazi do potpunog izlaganja dijelova modela. Stupnjevi eksplozije ne moraju se obavljati slijedno i u potpunosti ovise o interakciji korisnika.

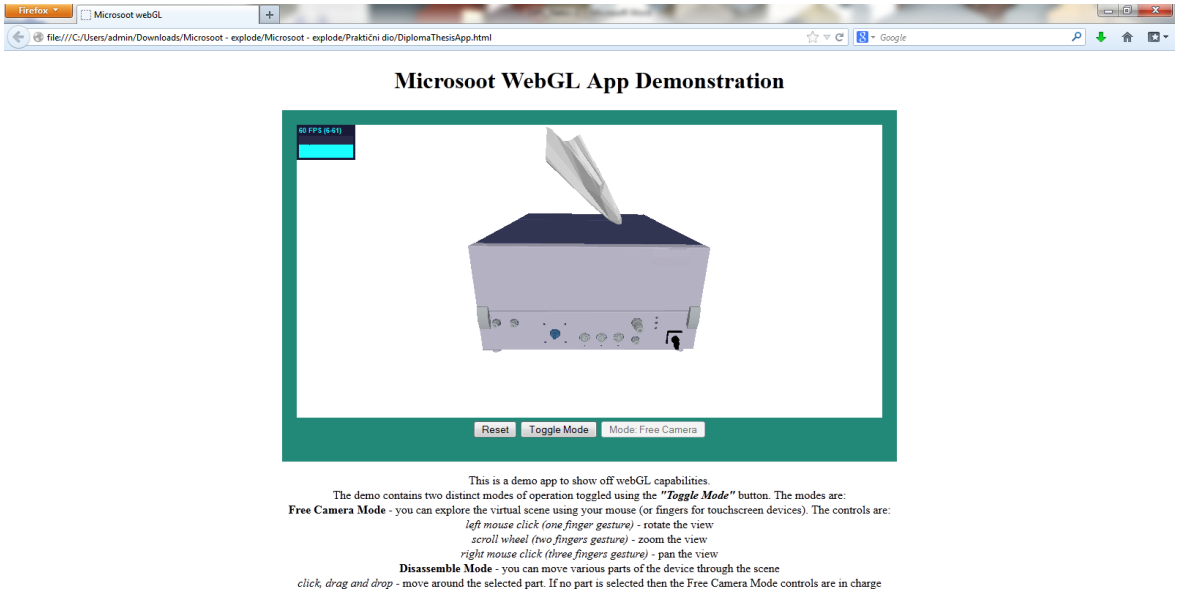
Dijelovi modela se razdvajaju u jednom od šest glavnih kanonskih smjerova, u smjeru pripadnih osi trodimenzionalnog koordinatnog sustava. Kompaktnost i vidljivost dijelova osigurana je odabirom prikladnih veličina pojedinih razmaka. Tijekom stupnjevanja vidljivost dijelova je ograničenog trajanja, prvenstveno radi bolje kompaktnosti. Naime, nakon početnog razdvajanja u narednom stupnju eksplozije ukida se vidljivost već istraženih dijelova. Princip eksplozije nastoji otvoriti model, tj. razdvojiti dijelove spremnika i izložiti njegove sadržane dijelove. Potom dijelovi spremnika prestaju biti vidljivi i postupak otvaranja modela se nastavlja. Nedostatak hijerarhije kompenziraju navedene skupine. Stoga je slijed eksplozije modela moguće okarakterizirati kao hijerarhijski.

Skupine su implementirane pomoću varijabli nizova (vektora), i kao takve pogodne za pretraživanje. Dijelovi skupina su JavaScript objekti, a definirani su svojim

imenom i koordinatama (pozicijama) koje zauzimaju unutar svoje rešetke (*eng. mesh*). Pozicioniranje dijelova unutar skupina odvija se prilikom inicijalizacije aplikacije.

Animacija pokretanja implementirana je pomoću biblioteke SimpleTween[28]. Za svaku animaciju stvara se zaseban *Tween* objekt. Za *ease* funkciju odabrano je jednoliko pravocrtno gibanje – *linear*. Korištena je sekvenca naredbi, čime se animacija ujedno i pokreće. Primjerice, animacija pomicanja dijela na poziciju (0,0,3) unutra 200 milisekundi definira se sekvencom naredbi:

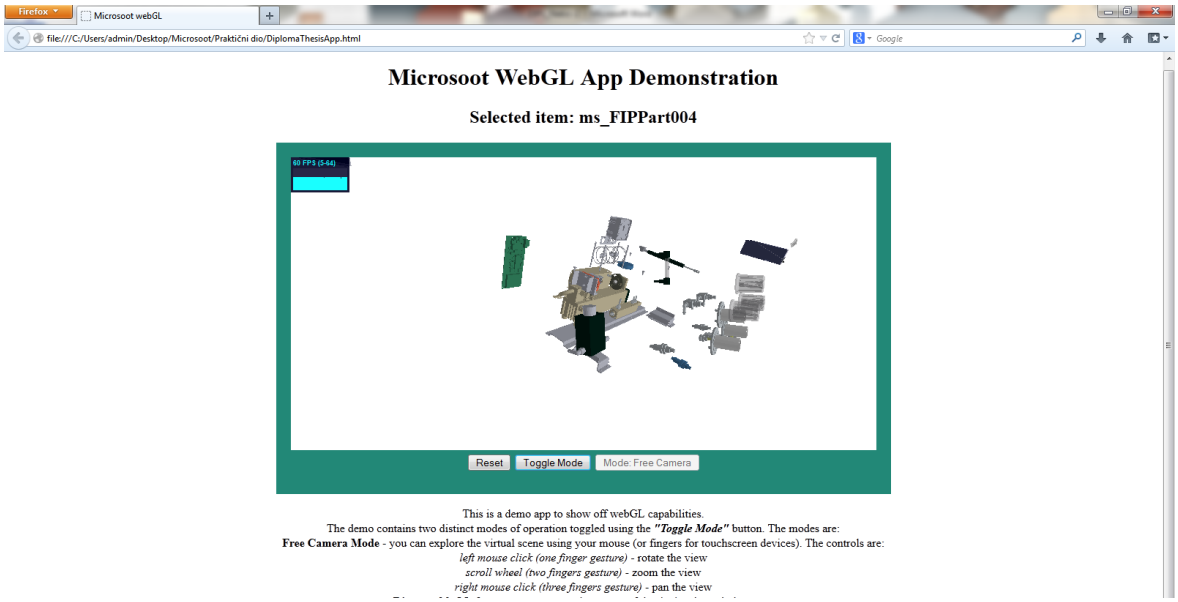
```
new TWEEN.Tween(objects[i].position).to( {x:0, y:0, z:3}, 200).start();
```



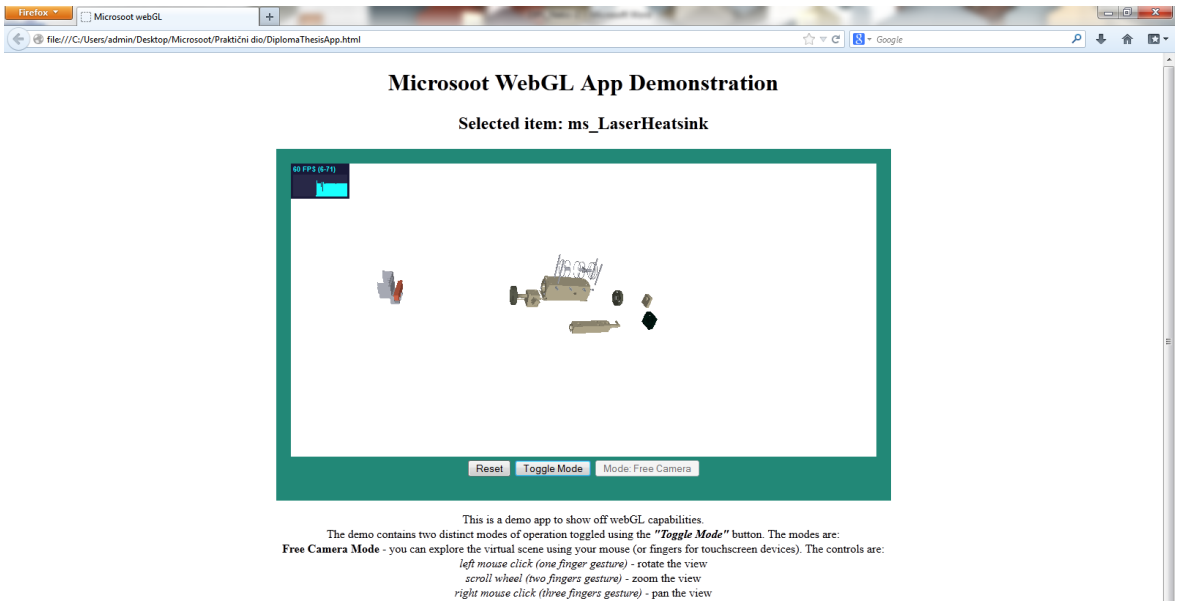

**Slika 4. Model Microsoot**


**Slika 5. Eksplozija uvjetovana odabirom dijela „ms_LidUpper“**


**Slika 6. Eksplozija uvjetovana odabirom dijela „ms_LidMid“**

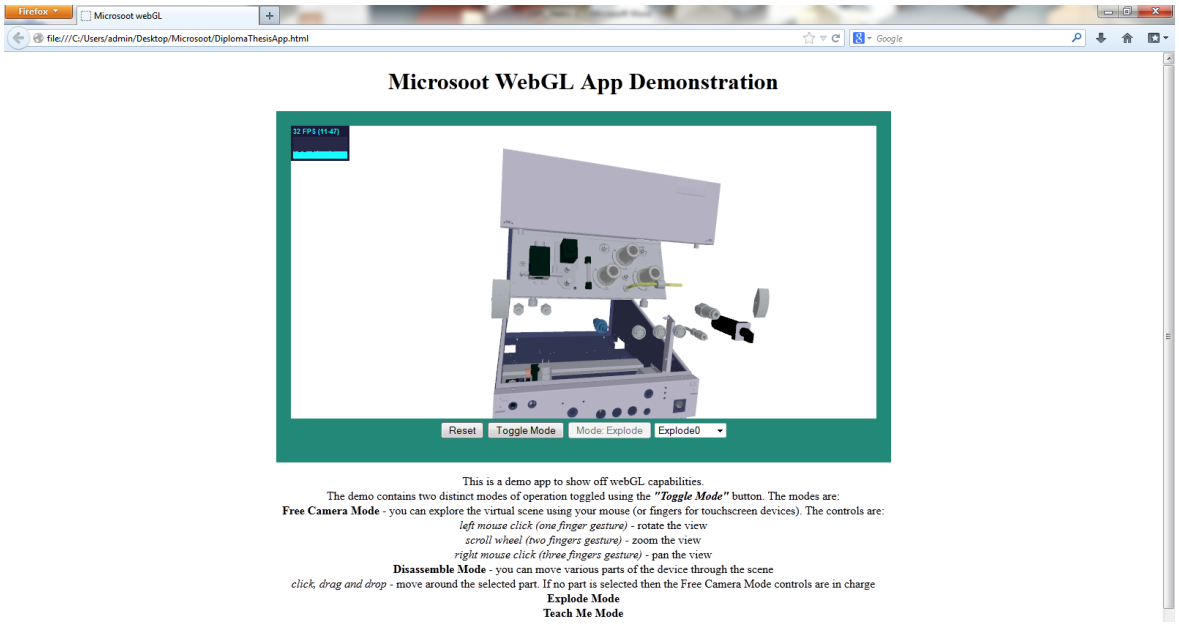



**Slika 7. Eksplozija uvjetovana odabirom dijela „ms_FIPPart004“**

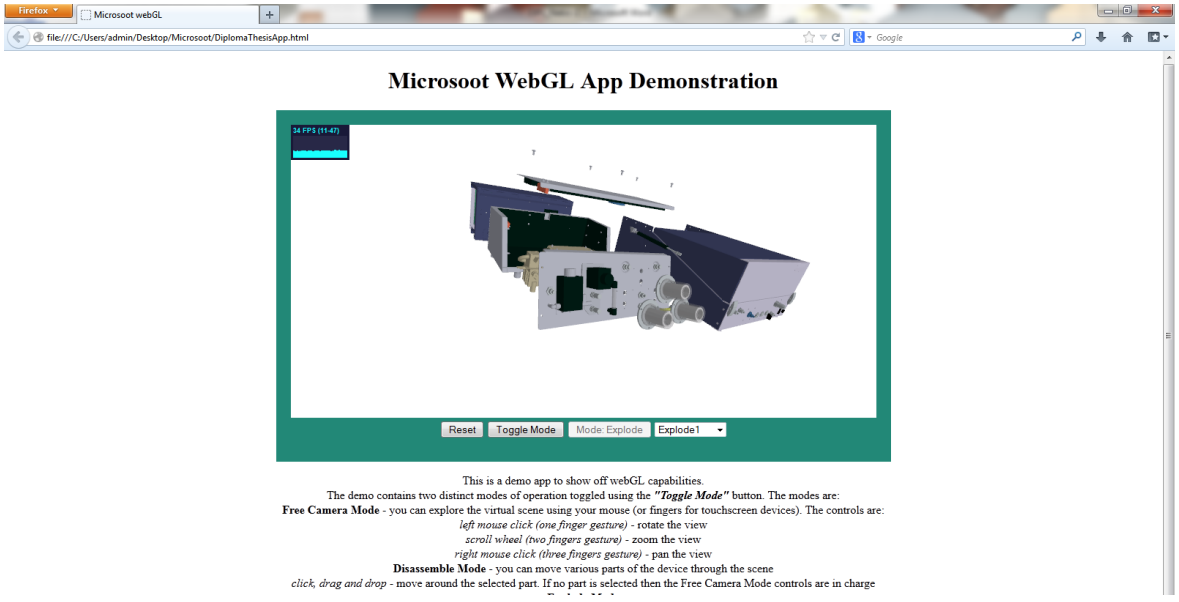



**Slika 8. Eksplozija uvjetovana odabirom dijela „ms_LaserHeatsink“**

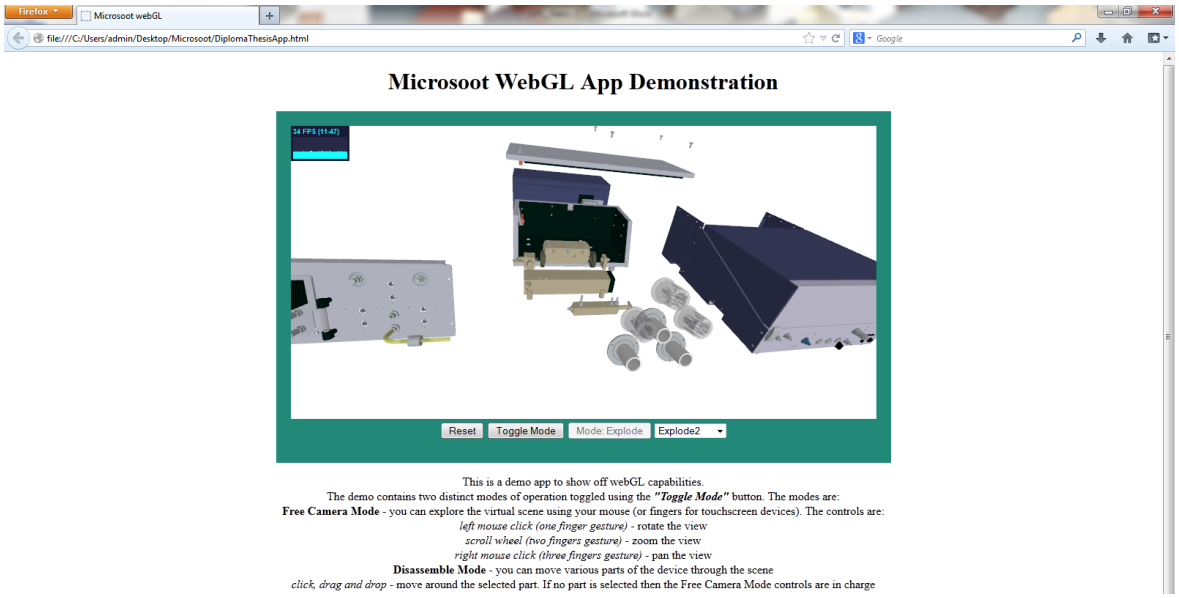
Implementirana je i verzija upravljanja eksplozijom pomoću izbornika. Podržane su četiri razine eksplozije.



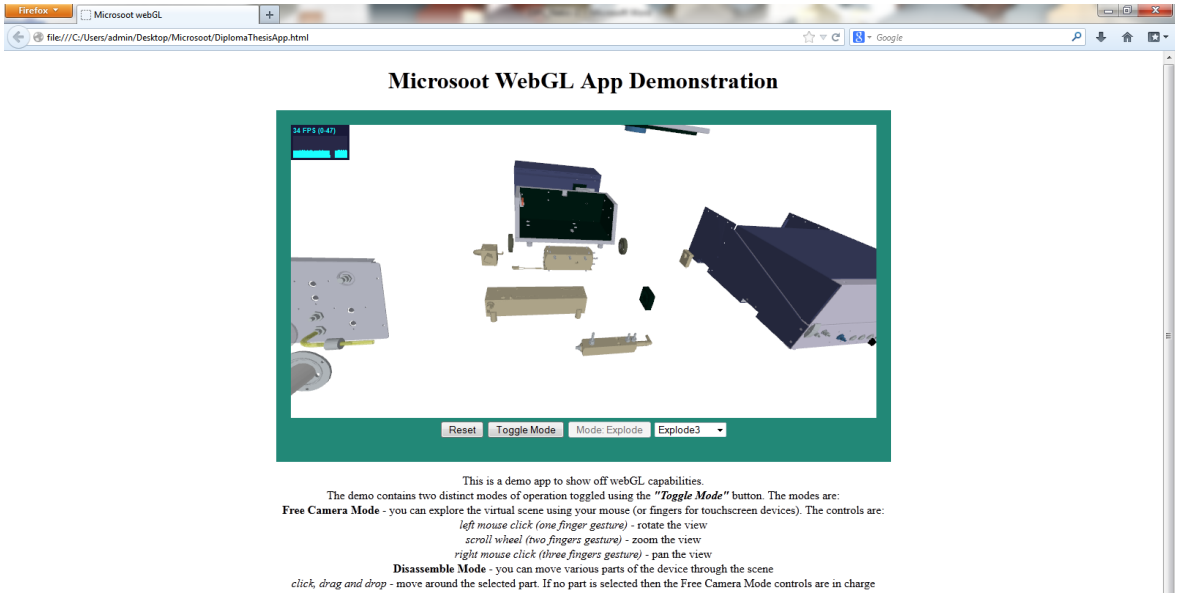

**Slika 9. Eksplozija "Explode0"**




**Slika 10. Eksplozija "Explode1"**

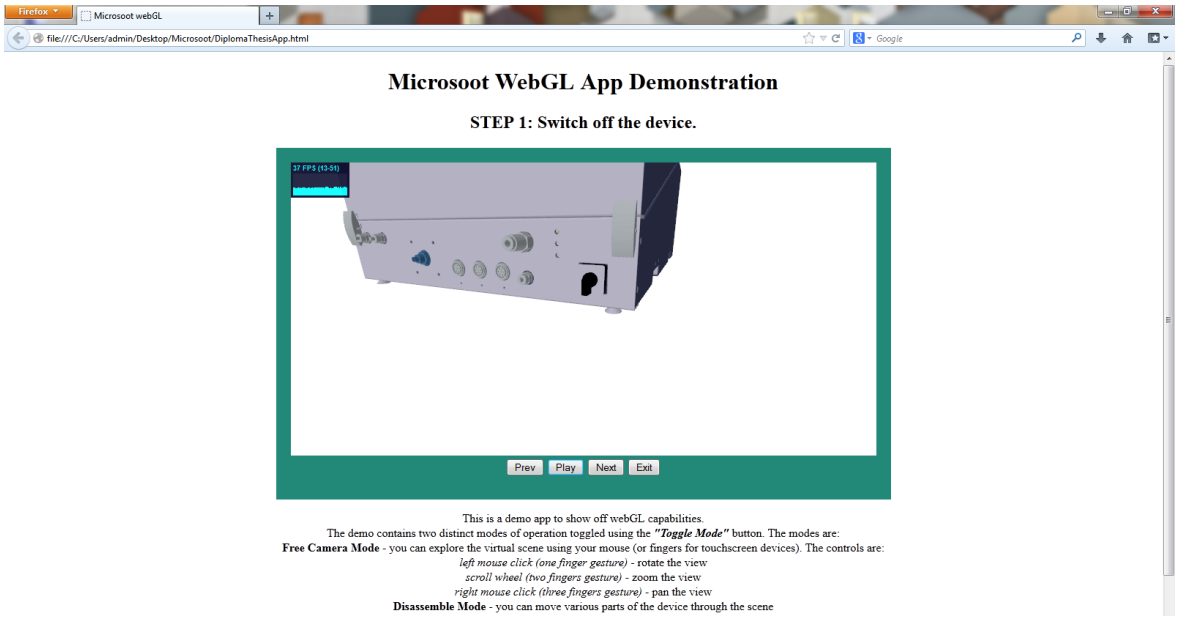



**Slika 11. Eksplozija "Explode2"**




**Slika 12. Eksplozija "Explode3"**

Osim eksplozijskog dijagrama implementirana je i edukacijska animacija interakcije s uređajem, prema dobivenom predlošku.




**Slika 13. Isječak iz animacije – sklopka**


**Slika 14. Isječak iz animacije - otvaranje uređaja**

# 10. Zaključak

Najnoviji standard za označavanje sadržaja koji se prikazuju u Web preglednicima – HTML5 unosi inovacije prilagođene suvremenoj tehnologiji. Osim novih elemenata, kojima je moguće intuitivnije definirati strukturu dokumenta za prikaz u Web preglednicima, karakteriziraju ga mehanizmi pohrane podataka i ostvarivanje komunikacije između klijenta i poslužitelja. Grafički se objekti prikazuju na HTML5-ovom elementu platno. HTML5 koristi vektorsku grafiku, tj. objekti se prikazuju pomoću grafičkih primitiva.

Jednostavnije je grafičke aplikacije moguće implementirati koristeći isključivo dostupne metode unutar elementa platno. Za razvijanje složenijih grafičkih aplikacija koristi se aplikacijsko programsko sučelje - WebGL. Njegove knjižnice pružaju potporu za razvoj grafičkih aplikacija, koje je moguće izvoditi na različitim uređajima – računalima, tabletima i pametnim telefonima.

Biblioteka Three pojednostavljuje korištanje WebGL-a, referencirajući se na njegove funkcije. Sadrži karakteristične objekte za definiranje postavki scene i ostvarivanje iscrtavanja, kao i učitavanje i ophođenje s objektima unutar prikaza. Također, pojednostavljuje ostvarivanje interakcije s mišem pomoću predefiniranih funkcija i varijabli za praćenje izmjena i pripadno iscrtavanje.

Biblioteka Tween pruža potporu implementaciji animacije objekata. Glavna namjena biblioteke Tween je ostvarivanje posredne integracije, odnosno kontinuiranog prijelaza između dvaju vrijednosti određenog parametra. Numerički parametar, čija vrijednost kontinurano prelazi iz početne u konačnu, moguće je proizvoljno odabrati.

Eksplozijski dijagrami se upotrebljavaju za prikaz složene strukture objekta. Prilikom definiranja rastavljanja objekta, nužno je osigurati kompaktnost i vidljivost dijelova, te spriječiti preklapanje. Općenito redoslijed eksplozije dijelova slijedi hijerarhiju modela ili prostorni raspored dijelova. Poželjno je koristiti dinamičke eksplozijske dijagrame i omogućiti interakciju.

Ana Nekić

# 11. Literatura


[1] HTML5 Introduction, *HTML5:HTML5 Introduction*,
http://www.w3schools.com/html/html5_intro.asp, 25.03.2013.
[2] HTML5 New Elements, *HTML5:HTML5 New Elements*,
http://www.w3schools.com/html/html5_intro.asp, 25.03.2013.
[3] HTML5 Canvas, *HTML5:HTML5 Canvas*,
http://www.w3schools.com/html/html5_canvas.asp, 25.03.2013.
[4] HTML5 Video, *HTML5:Video*,
http://www.w3schools.com/html/html5_video.asp, 25.03.2013.
[5] HTML5 Audio, *HTML5:Audio*,
http://www.w3schools.com/html/html5_audio.asp, 25.03.2013.
[6] HTML5 Drag nad Drop, *HTML5:Drag nad Drop*,
http://www.w3schools.com/html/html5_draganddrop.asp, 25.03.2013.
[7] HTML5 Geolocation, *HTML5:Geolocation*,
http://www.w3schools.com/html/html5_geolocation.asp, 25.03.2013.
[8] HTML5 Form Elements, *HTML5:New Form Elements*,
http://www.w3schools.com/html/html5_form_elements.asp, 25.03.2013.
[9] HTML5 Semantic Elements, *HTML5:New Semantic Elements*,
http://www.w3schools.com/html/html5_semantic_elements.asp, 25.03.2013.
[10] HTML5 SVG, *HTML5:Scalable Vector Graphics*,
http://www.w3schools.com/html/html5_svg.asp, 26.03.2013.
[11] HTML5 Input Types, *HTML5:Input Types*,
http://www.w3schools.com/html/html5_form_input_types.asp, 25.03.2013.
[12] HTML5 Web Storage, *HTML5:Web Storage*,
http://www.w3schools.com/html/html5_webstorage.asp, 26.03.2013.
[13] HTML5 Application Cache, *HTML5:Application Cache*,
http://www.w3schools.com/html/html5_app_cache.asp, 26.03.2013.
[14] HTML5 Server Sent Events, *HTML5:Server Sent Events*,
http://www.w3schools.com/html/html5_serversentevents.asp, 26.03.2013.
[15] HTML5 Web Workers, *HTML5:Web Workers*,
http://www.w3schools.com/html/html5_webworkers.asp, 26.03.2013.

[16] HTML, *HTML Living Standard,* http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html, 27.03.2013.
[17] McCormic, C., WebIDL, *WebIDL,* http://www.w3.org/TR/2012/CR-WebIDL-20120419/, 29.03.2013.
[18] Lie, H., Atkins, T. CSS Values and Units Module, 04.04.2013., http://www.w3.org/TR/css3-values/, 10.04.2013.
[19] Garci, D., Scan COnversion Distillation, *Chapter 3: Scan Conversion Distillation*, http://www.cs.berkeley.edu/~ddgarcia/cs184/r3/, 10.04.2013.
[20] RGraph, *An example of HTML5 hit region,* http://www.rgraph.net/blog/2013/january/html5-canvas-hit-regions.html, 10.04.2013.
[21] Marrin, C., WebGL Specification, 21.03.2013., *WebGL Specification*, https://www.khronos.org/registry/webgl/specs/1.0/, 20.04.2013.
[22] Robie, J., Document Object Model, *What si Document Object Model,* http://www.w3.org/TR/REC-DOM-Level-1/introduction.html, 10.04.2013.
[23] GitHub, *Three.js*, https://github.com/mrdoob/three.js/, 28.03.2013.
[24] ThreeJs, *ThreeJS*, http://threejs.org/, 25.03.2013.
[25] Learning Three.js, *Learning Three.js*, http://learningthreejs.com/, 25.03.2013.
[26] Lewis, P., Aerotwist, *Getting Started with Three.js*, http://www.aerotwist.com/tutorials/getting-started-with-three-js/ ,27.03.2013.
[27] Jacobs, D., Painter's Algorithm, *Painters Algorithm,* http://www.cs.umd.edu/~djacobs/CMSC427/VisibilityNonDiscrete.pdf, 10.04.2013.
[28] GitHub, *Tween.js*, https://github.com/sole/tween.js/, 10.04.2013.
[28] TweenJS, *Tween.js for Smooth Animation,* 17.08.2011., http://learningthreejs.com/blog/2011/08/17/tweenjs-for-smooth-animation/, 10.04.2013.
[29] Meegerman, P., Javascript Motion Tween, *JavaScript Animation Engine,* 28.01.2007., http://jstween.blogspot.com/, 10.04.2013.
[31] Kalkofen, D., Tatzgern M., Schmalstieg D. Explosion Diagrams in Augmented Reality, Proceedings of IEEE Virtual Reality, 2009.

[32] Li, W., Agrawala, M., Curless, B., Salesin, D. Automated Generation of Interactive 3D Exploded View Diagrams, SIGGRAPH 2008, Los Angeles, USA, 2008.
[33] Github, Three.js, *Three.js Blender Import Export,* https://github.com/mrdoob/three.js/tree/master/utils/exporters/blender, 06.04.2013.
[34] HTML5 Canvas, *HTML5 Canvas Tutorials,* http://www.html5canvastutorials.com/, 10.04.2013.

## 12. Popis slika

# Grafički objekti u Web preglednicima

## 13. Sažetak

U diplomskom radu je dan pregled značajki standarda HTML5 - najnovije inačice jezika za označavanje sadržaja u Web preglednicima. Grafički se objekti prikazuju na HTML5-ovom elementu platno, koji je detaljnije opisan. WebGL predstavlja aplikacijsko programsko sučelje za implementiranje grafičkih aplikacija, koje koriste HTML5 element platno. Biblioteka Three pojednostavljuje korištenje WebGL-ovih metoda. Pomoću biblioteke Three moguće je jednostavno definirati postavljanje scene i učitavanje grafičkih objekata, te interakciju s mišem. Biblioteka Tween koristi se za implementaciju animacije grafičkih objekata. Namjena biblioteke Tween je posredna integracija, tj. ostvarivanje kontinuiranog prijelaza između dvaju vrijednosti. U radu su opisane značajke eksplozijskih dijagrama. Edukativna aplikacija je implementirana u jeziku JavaScript.

**Ključne riječi** : HTML5, platno, WebGL, biblioteka Three, biblioteka Tween, eksplozijski dijagrami, JavaScript

# Graphic objects in Web browsers

## 14. Abstract

The Master Thesis gives an overview of HTML5 features. HTML5 is the latest version of the markup language for content displayed in Web browsers. Graphic objects are displayed on HTML5's canvas element, which is described in detail. WebGL is the application programming interface for implementing graphical applications using HTML5's canvas element. Three library simplifies the use of WebGL methods. Three library can easily define and load a scene of graphic objects and interact with the mouse. Tween library is used for animation. It implements *indirect integration*, i.e. the realization of continuous transition between two values. This paper describes the features of explosive diagrams. Educational application is implemented in JavaScript programming language.

**Keywords** : HTML5, canvas, WebGL, Three library, Tween library, Explosion diagrams, JavaScript