SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 3195.

RAČUNALNA ANIMACIJA NA WEBU

Ana Marija Komar

Zagreb, lipanj 2013.

Sadržaj

1 Uvod

Računalna grafika do danas je dosta napredovala i postoji mnogo tehnika kojima se sadržaj na ekranu može učiniti zanimljivim. Editori, video isječci, programi za crtanje i bojanje, i mnogi drugi rezultiraju slikom na ekranu. Također, internet i web tehnologije brzo napreduju, pa svaki dan raste i broj načina na koji određeni sadržaj može biti prikazan. Dok su u početku ljudi bili sasvim zadovoljni običnim tekstom i pokojom slikom na web stranicama, danas očekivanja iz dana u dan sve više rastu.

Prvu web stranicu napravio je 1981. godine Tim Berners-Lee. Ona uopće nije sadržavala nikakve slike. U veljači 1993. godine prvi puta je predstavljen svim web programerima dobro poznati tag <*img*>. Danas svaka stranica sadrži mnoštvo multimedijalnih sadržaja, od kojih su animacije jedan važan dio. Upravo one privlače pažnju korisnika. Više od četvrtine ljudskog mozga je uključena u interakciju s vizualnim signalima, stoga ljudi vole pokret i vole gledati kako se nešto miče. Prve animacije na webu pojavile su se 1987. s nastankom GIF slika.

Animacije mogu biti 2D i 3D animacije. Vizualizirati nešto u 3D znači definirati stvari u trodimenzijskom svijetu. U polju 3D grafike matematički se definiraju objekti, materijali, svjetlo i kamera, i oni se svi zajedno koriste da bi se slika prikazala na 2D ekranu. Monitor postaje prozor u svijet koji je stvoren.

3D web tehnologija otvara vrata mnogim novim zanimljivim sadržajima na stranicama. Jedan od načina koji se već koristi i nastoji se da postane standard jest mogućnost da se proizvod ponudi u 3D-u na samoj stranici, kako bi ga korisnici mogli pregledavati kao u stvarnosti, iz svakog kuta. Osim toga, 3D tehnologija može dodati novi stupanj učinkovitosti i prezentacije sadržaja kod raznih edukacijskih web stranica, u kojima se pomoću nje mogu simulirati i prikazati razni teoremi, primjeri i slično.

Uz 3D animacije na webu često se veže i pojam interaktivnost. Interaktivno znači da se može konstanto utjecati na ono što vidimo na ekranu. Video igre nam daju očiti primjer. Ako u igri utrka kliknemo npr. na strelicu prema lijevo idemo u

tom smjeru. U usporedbi s *DVD playerom*, za interakciju se koriste minimumi poput *fast forward*, *stop* itd., ali na niti jedan način se ne utječe na svijet. Interaktivni 3D znači da akcije korisnika utječu na nešto u 3D svijetu. Na primjer, pomakne se kamera prema objektu u 3D svijetu, dobije se drugačiji utjecaj na objekte i na osvjetljenje u tom svijetu.

Takvu se grafiku sasvim sigurno može vidjeti u igrama i u posebnim efektima u filmovima. I upravo to su dvije očite uporabe tehnike za 3D vizualiziranje. Arhitekti često stvaraju prezentacije u 3D grafici za svoje dobro, a i dobro klijenata. Inženjeri se koriste time da riješe probleme dizajna. Doktori koriste 3D rekonstrukcije zajedno s medicinskim uređajima kako bi razumjeli probleme pacijenta, čak i odvjetnici koriste 3D grafiku kako bi rekonstruirali događaje i prikazali tijek nesreće. Postoji još mnogo uporaba 3D vizualiziranja koji utječu na to kako svijet funkcionira.

U današnje vrijeme posvuda na internetu se mogu pronaći animirani oblici. Postoji mnogo tehnologija koje web dizajneri mogu iskoristiti da bi napravili animacije, a neki od njih su [1] :

- animirani GIF-ovi
- dinamički HTML
- *Java*
- *Shockwave* and *Flash*

Dok je 2D grafika na webu dobro razvijena, 3D grafika je u razvoju i iz dana u dan sve više napreduje. Cilj je ostvariti 3D grafiku na webu kao standardni dio svakog web preglednika.

U ovom radu proučit ću tehnologije koje omogućuju ostvarivanje računalnih animacija u web preglednicima bez uporabe dodataka (engl. *plug-in*). Posebno ću pažnju obratiti na 3D animacije i parametre kojima se kontroliraju 3D animacije. Obraditi ću mogućnostima animiranja u HTML5-u i animacije putem CSS-a.

2 Animacije i tehnologije

Animacija je brzo prikazivanje niza 2D ili 3D slika modela postavljenih tako da stvaraju privid pokreta. Ostvaruje se prikazom nacrtanih objekata u različitim položajima u svakom okviru pa izgleda kao da se objekt miče kad se okviri izmjenjuju određenom brzinom.

Animacija je moguća zbog tromosti oka. Objekt kojeg je oko vidjelo ostaje zapamćen još nekoliko trenutaka nakon gledanja.

Slika 1. Primjer 4 okvira animacije

Filmovi se snimaju s minimalno 24 okvira po sekundi (engl. *fps - frames per second*), a brzina se duplicira jer se svjetlost dva puta propušta kroz objektiv filmskog projektora što uzrokuje brzinu od 48 puta u sekundi. Animacije na računalu kontroliraju se pomoću vremenske osi, definira se ukupno trajanje animacije, te trenutak kada se koji okvir pojavljuje, odnosno postavljanjem broja okvira animacije (*frame rate*). Slika 1. prikazuje 4 okvira animacije i 4 različita položaja lika kojeg želimo animirati. Izmjenom 5 sličica u sekundi dobije se dosta dobar privid kretanja.

Osnovni tipovi animacije su:

- animacija po stazi - objekt se pomiče po putanji, ali se ne mijenja njegov oblik nego samo položaj
- animacija s različitim kadrovima - crta se niz crteža animacije u kojima se postepeno prelazi iz jednog kadra u drugi
- animacija preobražavanjem - računalni program generira međuslike pa se jedna slika pretvara u drugi

Jedan od načina da to programski ostvarimo na webu je da učitavamo sliku po sliku i izmjenjujemo ih u željenom vremenu. Kako bi se takvi podaci o slici komprimirali podaci koriste se GIF datoteke. Prvi GIF predstavljen je 1987. godine. To je bila prva animacija na webu, a i u današnje vrijeme je to jedan od najpopularnijih načina prikaza animacije na webu.

Prednost GIF-a je to što je s njim jednostavno raditi i automatski se prepoznaje u svim web preglednicima [1]. Danas postoji mnoštvo besplatnih i jednostavnih programa koji omogućuju izradu GIF-ova. Glavni nedostatak je taj što animacija mora biti izrazito jednostavna da bi se veličina datoteke mogla što više smanjiti jer je svaki okvir jedna cijela slika. Stoga su animirani GIF-ovi podosta ograničeni, a već kod brzine od 20 okvira u sekundi dolazi do problema kod učitavanja.

Jedan od načina kako riješiti taj problem je da se eliminiraju zasebni okviri, odnosno da se uzme jedna slika i pomiče se po ekranu. U početku web stranice su bile većinom statične (nakon što su se učitale, više se nisu mijenjale). Evolucijom interneta statična stranica je postala previše ograničavajuća. Web programeri željeli su dodati dinamički sadržaj na svoje web stranice. Tako se pojavio pojam DHTML (engl*. Dynamic hypertext markup language*) [1], odnosno dinamički HTML koji se ostvaruje pomoću skriptnih jezika, poput Javascript-a, koji pristupaju objektu na stranici (engl. DOM - *Document Object Model*). Ovaj način je također dosta ograničen jer omogućuje samo izmjenu ili pomak određenog elementa (objekta) na stranici (npr. pomakom miša preko teksta promijeni se boja teksta).

Pojavom programskog jezika *Jave* 1995. omogućeno je da se na webu prikažu i kompleksnije animacije. Male aplikacije pisane u Javi koje se mogu izvoziti u obliku bytecoda, poznate pod nazivom *Java appleti*, mogu se prikazati u zasebnom okviru unutar web stranice. *Java appleti* su prikladni za stvaranje interaktivnih animacija koji se kombiniraju uz ostali sadržaj na web stranici. Prednost Java appleta, kao dodatka web pregledniku je to da se može koristiti na bilo kojem operacijskom sustavu i pruža različite mogućnosti pri izradi malih aplikacija. Uz *Java applete* među najpoznatije dodatke (engl. *plug-in*) web preglednicima su *Flash* i *Schockwave*.

Shockwave je originalno bio napravljen za kreiranje dinamičkih datoteka za *CD-ROM*-ove, dok je *Flash* napravljen za uporabu na webu [1]. *Flash* je jeftiniji program, dok je *Schockwave* prikladniji za neke složenije animacije. Iako su oba dosta jednostavna za upotrebu, nisu previše prikladni za 3D animacije.

Donedavno velik problem bili su spora računala i spore veze prema Internetu. Za učitavanje 3D sadržaja bilo je potrebno previše vremena, a jednom kad se učitalo sporo se prikazivalo. S razvojem bržih procesora i brže veze prema internetu pojavila se i mogućnost razvoja 3D weba. 2006. godine pojavio se prvi *Canvas* 3D prototip, a u ožujku 2011. je objavljena vezija 1.0 WebGL specifikacije za koji se danas nastoji da postane prihvaćen kao dio web standarda.

3 CSS i animacije

CSS (engl. Cascading Style Sheets) je skup pravila koja govore web pregledniku kako prikazati određenu HTML oznaku. Prvenstveno je napravljen kako bi se odvojio dizajn web stranice od sadržaja dokumenta. CSS3 je najnovija verzija CSS-a.

Korištenjem CSS atributa HTML elementi mogu biti pomaknuti unutar web stranice, te se može utjecati na njihovu visinu, širinu, boju i mnoštvo drugih efekata. Novost u CSS3-u je atribut *transform* pomoću kojeg se elementi mogu skalirati, pomicati bez uporabe pozicioniranja te rotirati. Do uvođenja CSS3-a animacije su bile moguće korištenjem *JavaScript*-a, a sada su animacije moguće i kada je *JavaScript* onemogućen.

Glavni nedostatak vizualnih efekata u CSS3-u je što u različitim web-preglednicima treba iskoristiti prefikse, ovisno o tome koji atribut se upotrebljava pa je istu liniju koda potrebno upisati više puta. Slika 2. pokazuje dio tablice koja pokazuje podržanost novih atributa u CSS3-u u web preglednicima.

SVG (engl. *Scalable Vector Graphics*) je XML jezik za prikazivanje dvo-dimenzionalne vektorske grafike, bilo nepomične ili animirane. Animacije u CSS3-u su moguće zbog takvog načina zapisa podataka. Mogućnosti animiranja putem CSS-a su također dosta ograničene, pogotovo kada je u pitanju 3D grafika.

CSS3 možemo podijeliti u module:

- selektore
- kutije (engl. *box models*)
- pozadina i granice
- tekstualni efekti
- 2D/ 3D transformacije
- animacije
- izgled s višestrukim stupcima
- korisničko sučelje

The table below lists the new CSS3 properties and their browser support:

Property	IE	Firefox	Chrome	Safari	Opera
alignment-adjust					
alignment-baseline					
@keyframes	10	16	-webkit-	-webkit-	12.1
animation	10	16	-webkit-	-webkit-	12.1
animation-name	10	16	-webkit-	-webkit-	12.1
animation-duration	10	16	-webkit-	-webkit-	12.1
animation-timing-function	10	16	-webkit-	-webkit-	12.1
animation-delay	10	16	-webkit-	-webkit-	12.1
animation-iteration-count	10	16	-webkit-	-webkit-	12.1
animation-direction	10	16	-webkit-	-webkit-	12.1
animation-play-state	10	16	-webkit-	-webkit-	12.1
appearance		-moz- 3	-webkit-	-webkit-	
backface-visibility	-ms- 10	-moz-	-webkit-	-webkit-	
background-clip	9	4	4	5	10.5
background-origin	9	4	4	5	10.5
background-size	9	4	4	5	10.5
baseline-shift					
bookmark-label					
bookmark-level					
bookmark-target					
border-bottom-left-radius	9	4	5	5	10.5
border-bottom-right-radius	9	4	5	5	10.5
border-image		15	16	6	11
border-image-outset					

Slika 2. Dio tablice koja prikazuje podrživost web preglednika za atribute CSS3-a

3.1 Transformacije

Transformacija je efekt koji utječe na element tako da mijenja njegov oblik, veličinu ili poziciju. CSS3 transformacije omogućuju pomicanje, skaliranje, okretanje i rastezanje elemenata [2].

CSS vizualni model opisuje koordinatni sustav u kojem je svaki element pozicioniran. Pozicije i veličine elemenata u ovom kooridnatnom sustavu izražavaju se u pikselima počevši od gornjeg lijevog kuta. Koordinarni sustav se može modificirati pomoću svojstva *transform*. Atribut *transform* podržan je u

Operi, *Mozilli*, *Internet Exploreru* (prefiks -ms-), te *Google Chrome*-u i *Safari*-ju uz prefiks -webkit-.

Dodatna svojstva čine transformacije jednostavnijima i dopuštaju autoru da kontrolira interakciju ugniježđenih 3D transformacija [4].

- *Transform-origin* omogućuje pogodan način za kontroliranje koja je transformacija upotrebljena na kojem elementu.

- Svojstvo *perspective* omogućuje autoru da elementu napravi posebne elemente (djecu) 3D transformacijama tako da oni i dalje čine isti 3D prostor. Svojstvo *perspective-origin* omogućuje odrediti na koju je perspektivu to svojstvo upotrebljeno ne imajući utjecaj na ostale elemente.

- *Transform-style* omogućuje stvaranje hijerarhije 3D objekata na kojima su primijenjene 3D transformacije.

- *Backface-visibility* je svojstvo koje dolazi do izražaja kada se objekt okreće preko trodimenzionalnih transformacija i utječe na to da je zadnja strana vidljiva gledatelju. Vrlo često je poželjno sakriti zadnju stranu što se postiže atributom *hidden*.

3.1.1 2D transformacije

2D transformacije preko atributa *transform* omogućuju nam skaliranje, rotiranje i translaciju elemenata. Jednom elementu moguće je pridružiti više transformacija.

Sve promjene rade se uporabom matrice homogenog 2D prostora koja je sljedećeg oblika:

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

Funkcije koje omogućuju 2D transformacije:

- **matrix(a, b, c, d, e, f)**

  Opisuje 2D transformaciju u obliku transformacijske matrice koja sadrži 6 brojčanih elemenata.

- **translate(t_x [, t_y])**

  Opisuje 2D translaciju vektorom [t_x, t_y], gdje je t_x pomak u smjeru osi x (pozitivan smjer u desno), a t_y je opcionalan parametar koji opisuje pomak u smjeru osi y (pozitivan smjer prema dolje). Ukoliko se t_y ne navede njegova vrijednost je 0.

- **translateX(t_x), translateY(t_y)**

  Prva omogućuje pomak samo u smjeru x, a druga omogućuje pomak samo u smjeru y.

- **scale(s_x [, s_y])**

  Omogućuje primjernu 2D skaliranja vektorom [s_x , s_y]. Parametar s_y nije nužno navoditi, a u tom slučaju on poprima vrijednost jednaku prvom parametru.

- **scaleX(s_x), scaleY(s_y)**

  Prva omogućuje skaliranje po x osi, odnosno vektorom [s_x, 1], a druga skaliranje po y osi, odnosno vektorom [1, s_y].

- **rotate(α)**

  Opisuje 2D rotaciju za kut α upisan kao parametar. Rotacija je u smjeru kazaljke na satu (engl. CW clockwise), a upis parametra je oblika "90deg".

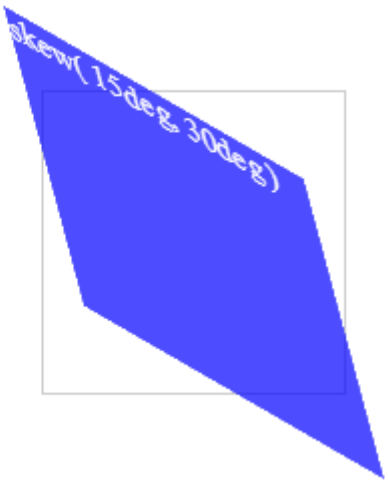
- **skew(α [, β])**

  Opisuje smik, odnosno uzdužnu transformaciju sa parametrima α i β za x i y os. Ako drugi parametar nije naveden njegova vrijednost je 0.

- **skewX(α), skewY(α)**
  Prva opisuje 2D uzdužnu transformaciju po x osi za dani kut α, a druga opisuje 2D uzdužnu transformaciju po y osi za dani kut α.

Sljedeći isječak koda pokazuje primjenu smika, a rezultat je prikazuje Slika 3. Kako bi se transformacija vidjela u svim preglednicima uz istu funkciju nužno je navesti potrebne prefikse. Uzdužna transformacija provodi se nad div elementom klase 'ploca' i to 15° po osi x i 30° po osi y.

```
<style media="screen">

  .container {
      width: 150px;
      height: 150px;
       border: 1px solid #CCC;
      -webkit-perspective: 600px;
      -moz-perspective: 600px;
      -o-perspective: 600px;
         perspective: 600px;
  }

    .ploca {
      width: 100%;
      height: 100%;
      position: absolute;
      opacity: 0.7;
      color: white;
      background: blue;
}

#smik .ploca {
 -webkit-transform: skew( 15deg, 30deg );
    -moz-transform: skew( 15deg, 30deg );
      -o-transform: skew( 15deg, 30deg );
         transform: skew( 15deg, 30deg );
}
</style>

<body>

      <div id="smik" class="container">
            <div class="ploca">skew( 15deg, 30deg )</div>
      </div>
</body>
```



Slika 3. Smik transformacija

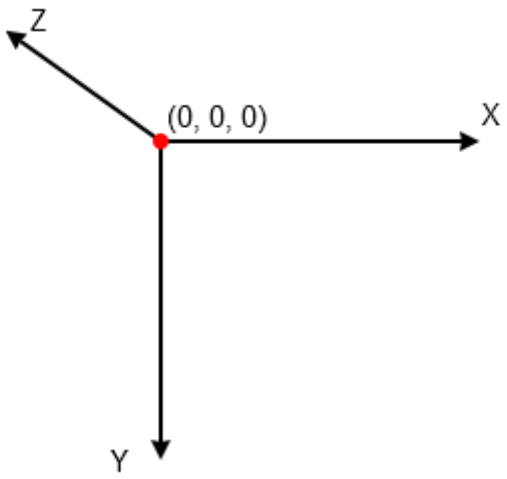
3.1.2 3D transformacije

Uporabom atributa *transform* stvara se lokalni koordinatni sustav za element na kojem je primjenjena transformacija. Preslikavanje mjesta u koordinatnom sustavu od kuda će element biti prikazan u lokalni koordinatni sustav određuje transformacijska matrica koju zovemo matrica transformacije pogleda. Transformacije su kumulativne, što znači da element ostvaruje svoj koordinatni sustav unutar koordinatnog sustava roditelja.

Koordinatni sustav se sastoji od 2 osi: X koja se povećava vodoravno u desno i Y koja se povećava okomito prema dolje. 3D transformacijske funkcije proširuju koordinatni sustav u 3 dimenzije, dodavši Z os koja je okomita na ravninu ekrana i raste prema gledatelju. Inicijalni koordinatni sustav je prikazan na Slici 4.

Transformacijska matrica računa se iz svojstava *transform* i *transform-origin* na sljedeći način:

1. Započinje se s matricom identiteta.
2. Translatira se za X, Y i Z vrijednosti iz atributa *transform-origin*.
3. Matrica se pomnoži po redu sa svakom transformacijskom funkcijom iz atributa *transform*.
4. Translatira se za negativne vrijednosti izračunate X, Y i Z iz atributa *transform-origin*.



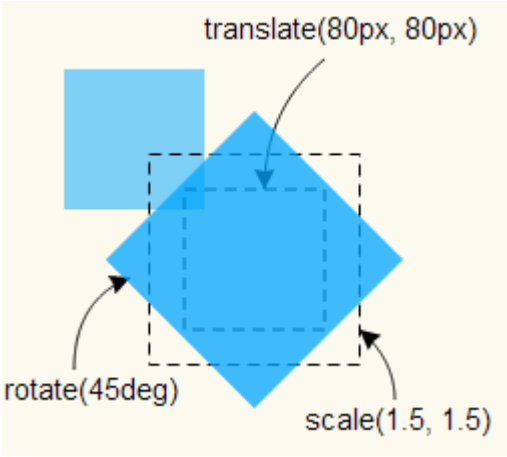
Slika 4. Početni koordinatni sustav

Ono što je također bitno primijetiti je da ako se transformacija primijeni na korijenski element, onda će imati utjecaj i na svu njegovu djecu.

Sljedeći primjer pokazuje primjene niza transformacija:

```
div {
    height: 100px; width: 100px;
    transform: translate(80px, 80px) scale(1.5) rotate(45deg);
}
```

Slika 5. pokazuje kako utječu transformacije na element. Visina div elementa je 100px, a širina 100px i redom se primijenjuju translacija za 80 px u desno i 80 piksela prema dolje, zatim skaliranje po x i po y osi za 1.5, te rotacija za 45°.

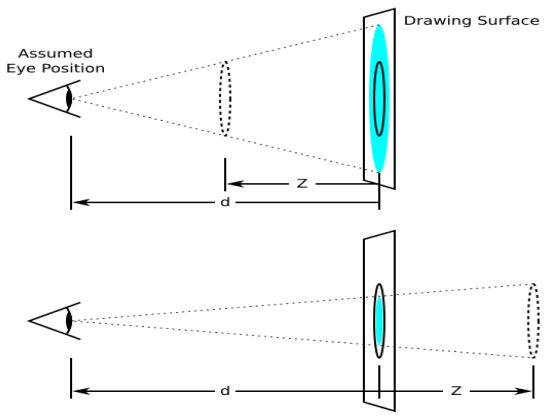


Slika 5. Utjecaj transformacija na element

3D prikazivanje

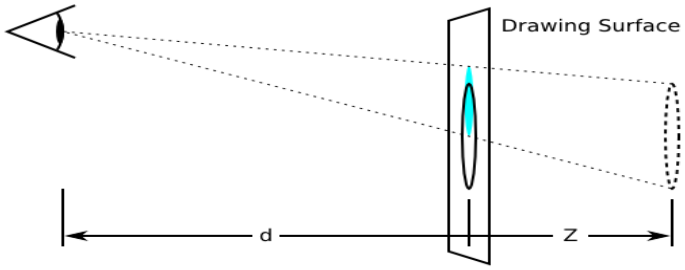
Trodimenzionalne transformacije najčešće rezultiraju s matricom čija je z komponenta različita od 0. Posljedica toga je da se z-os projicira na ravninu koja je drugačija od one koja sadrži taj element. To utječe na redoslijed prikaza elemenata i na njihovo međusobno ispreplitanje (engl. *front to back rendering order*). Ovo ponašanje ovisi o tome da li je element dio 3D prikaznog konteksta.

Svojstva *perspective* i *perspective-origin* mogu se iskoristiti kako bi se dodao osjećaj dubine prostora, odnosno da bi se element prikazao bliže ili dalje gledatelju. Skaliranje je proporcionalno vrijednosti d/(d-z), gdje je d vrijednost atributa *perspective*, odnosno udaljenost ravnine u kojoj se crta od pretpostavljenog položaja oka promatrača. Slika 6. pojašnjava ovaj odnos.



Slika 6. Korištenje atributa "perspective" za prikaz dubine

Atribut *perspective-origin* se odnosi na slučaj kada se uzima u obzir da pozicija oka promatrača nije u središtu elementa. Primjena je vidljiva kada se u ravnini prikaza prikazuje više od jednog elementa koji dijele zajedničku perspektivu. Utjecaj takvog pomaka prikazan je na Slici 7.



Slika 7. Utjecaj atributa "perspective-origin"

Samo elementi koji su dio 3D konteksta prikaza se mogu vidljivo preplitati s ostalim elementima tog konteksta. Ako na element primijenimo 3D transformaciju, ali on nije dio 3D konteksta prikaza, na element će biti primijenjena ta transformacija, ali on se neće isprepletati s ostalim elementima. U tom slučaju će rezultat biti isti takav kao da smo primijenili samo 2D transformacije.

Pozicija svakog elementa u 3D prostoru određena je zbirnom transformacijskom matricom u tom 3D prostoru kroz element koji ostvaruje 3D kontekst prikaza i sve elemente koji su dio tog konteksta.

Element ostvaruje i postaje dio 3D konteksta prikaza na sljedeći način:

- 3D kontekst prikaza ostvaruje transformacijski element čija vrijednost atributa *transform-style* je *preserve-3d* i koji sam po sebi nije dio 3D konteksta prikaza, ali sudjeluje u tom kontekstu.

- Element čija vrijednost atributa *transform-style* iznosi *preserve-3d* i koji je dio tog konteksta, proširuje postojeći 3D kontekst prikaza.

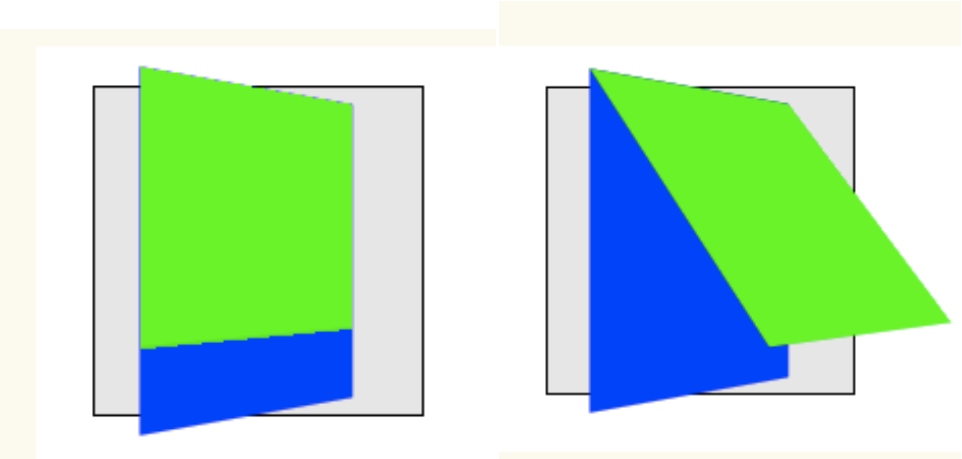
- Element sudjeluje u 3D kontekstu prikaza ukoliko blok koji ga sadrži ili ostvaruje ili proširuje 3D kontekst prikaza.

- Konačna vrijednost transformacije koja se koristi za prikaz 3D konteksta računa pomoć zbirne 3D transformacijske matrice na sljedeći način:

    1. Započinje se s matricom identiteta.
    2. Za svaki element koji je dio 3D konteksta prikaza:
        a. Pomnoži se zbirna matrica s perspektivnom matricom elementa. Perspektivna matrica računa se u odnosu na korijen, tj. u odnosu na element koji ostvaruje 3D kontekst prikaza.
        b. Na zbirnu matricu se upotrijebi translacija kao relativan pomak u odnosu na blok koji taj element sadrži.
        c. Zbirna matrica se pomnoži s transformacijskom matricom.

Sljedeći primjer pokazuje važnost svojstva *preserve-3d*. Plavi element ostvaruje 3D kontekst prikaza, a zeleni element je njegov dio. Slika 8. pokazuje plavi i zeleni element koji dijele isti 3D prostor. Sve transformacije koju su primijenjene su 3D transformacije. Lijevo je rezultat koji se dobije kada bi se zacrnjena linija koda izostavila, a desno je prikaz koji ostvaruje 3D dubinu i željeni rezultat isječka koda.

```
<style>
    div {
            height: 150px;
            width: 150px;
    }
    .container {
            perspective: 500px;
            border: 1px solid black;
    }
    .transformed {
            transform-style: preserve-3d;
            transform: rotateY(50deg);
            background-color: blue;
    }
    .child {
            transform-origin: top left;
            transform: rotateX(40deg);
            background-color: lime;
    }

</style>
```



Slika 8. Važnost navođenja atributa *preserve-3d*

Funkcije koje omogućuju 3d transformacije:

- **matrix3d(a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p)**
  Homogena matrica 4x4 sa 16 vrijednosti određuje 3D transformaciju.

- **translate3D(t_x, t_y, t_z)**
  Određuje 3D pomak u respektivno u smjeru x, y i z osi.

- **translateZ(t_z)**
  Određuje 3D translaciju vektorom [0,0,t_z], odnosno za iznos t_z u smjeru z osi.

- **scale3d(s_x, s_y, s_z)**
  Određuje operaciju skaliranja vektorom [s_x, s_y, s_z].

- **scaleZ(s_z)**
  Skaliranje objekta po osi z, odnosno vektorom [1, 1, s_z].

- **rotate3d(x, y, z, α)**
  Određuje 3D rotaciju za kut određen parametrom α oko usmjerenog vektora [x, y, z]. Ako je usmjereni vektor jednak [0, 0, 0] rotacija se neće primijeniti. Rotacija je usmjerena u smjeru kazaljke na satu.

- **rotateX(α)** = **rotate3d(1, 0, 0, α)**, **rotateY(α) = rotate3d(0, 1, 0, α)**, **rotateZ(α)** = **rotate3d(0, 0, 1, α)** = **rotate(α)**

- **perspective(d)**
  Određuje perspektivnu projekcijsku matricu. Ova matrica skalira vrijednosti u X i Y u odnosu na njihovu Z vrijednost, ukoliko je Z pozitivan vrijednosti se udaljavaju od ishodišta, a negativan Z približava ishodištu.
  Parametar d označava udaljenost ravnine s parametrom Z=0 od gledatelja. Manje vrijednosti rezultiraju elementom koji će biti spljošteniji. Vrijednost dubine mora biti veća od 0.

- **translateX(t_x), translateY(t_y), scaleX(s_x), scaleY(s_y)**
  Ponašaju se kao i u 2D slučaju.

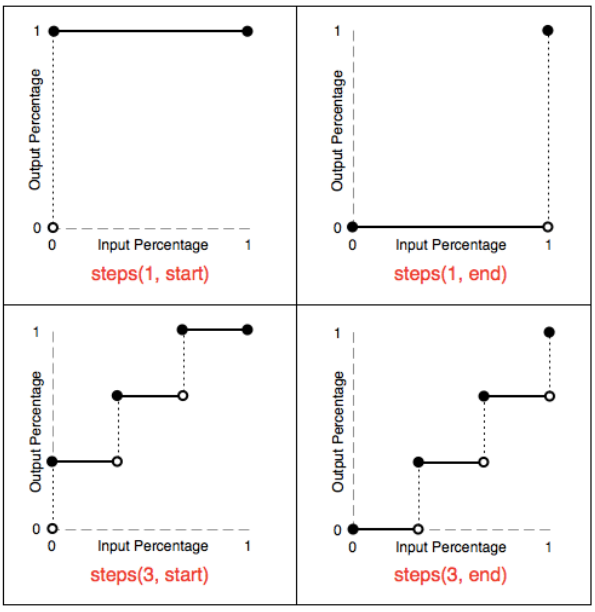
3.2 Animacije

3.2.1 CSS Tranzicije

Jedan od mogućih načina ostvarivanja animacija u CSS-u je pomoću tranzicija. Ova mogućnost je uvedena u CSS3 i njome možemo dodati efekte da se promjena odvija postupno jednog stila u drugi u određenom periodu [5].

Internet Explorer, Firefox, Chrome i *Opera* podržavaju *transition* atribut. U *Safari*-ju je ispred *transition* potrebno navesti prefiks -*webkit*-.

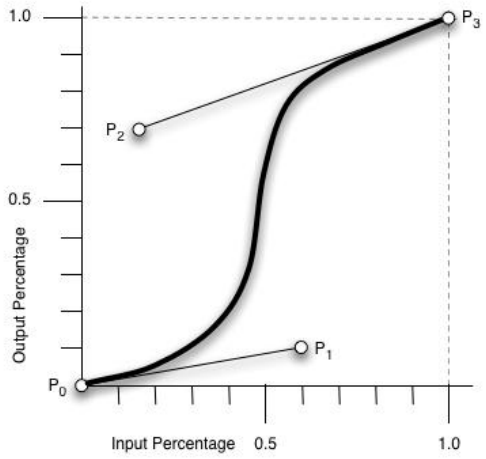
- **transition-property**
  Određuje na koje CSS svojstvo će se prijelaz primijeniti. Inicijalna vrijednost je *all*. Uz CSS svojstva moguće je navesti i sljedeće : *none*, *inherit*.
- **transition-duration**
  Određuje trajanje prijelaza. Podrazumijevana vrijednost je 0s.
- **transition-delay**
  Određuje vremenski odmak početka tranzicije.
- **transition-timing-function**
  Svojstvo koje opisuje kako će se vrijednosti izmjenjivati. Ono dopušta da se brzina mijenja tijekom trajanja prijelaza. Tranzicijske funkcije su definirane step-funkcijom ili na kubnom Bezierovom krivuljom.



Slika 9. Primjer step funkcije

Step funkcija

Step funkcija je određena brojem koji domenu dijeli na jednake intervale. Svaki podinterval je korak (engl. *step*) bliži krajnjem stanju. Funkcija također određuje i da li se promjena obavlja na početku ili na kraju intervala. Primjeri step funkcije su prikazani na Slici 9. Prikazane su 4 različite step funkcije. Gornji dio pokazuje kako izgleda step funkcija kada je interval jednak 1, odnosno nije podijeljen na intervale, a donji kada je funkcija podijeljena na 3 intervala. Lijeva strana prikazuje promjenu na početku, a desna na kraju.



Slika 10. Primjer kubne Bezierove krivulje i 4 kontrolne točke

Kubna Bezierova funkcija

Ukoliko ne želimo da intervali budu jednake duljine, odnosno ne želimo da se postotak promjene odvija periodično definirat ćemo kubnu Bezierovu krivulju. Kubna Bezierova krivulja je definirana s 4 kontrolne točke. Prva i zadnja točka uvijek označavaju početnu i krajnju poziciju/transformaciju, a *transition-timing-function* svojstvo određuje vrijednosti za točke P1 i P2. Slika 10. pokazuje kubnu Bezierovu krivulju.

Ovo su definicije vremenskih funkcija(engl. *timing functions*):

- **ease** ≡ cubic-bezier(0.25, 0.1, 0.25, 1)
- **linear** ≡ cubic-bezier(0, 0, 1, 1)
- **ease-in** ≡ cubic-bezier(0.42, 0, 1, 1)
- **ease-out** ≡ cubic-bezier(0, 0, 0.58, 1)
- **ease-in-out** ≡ cubic-bezier(0.42, 0, 0.58, 1)
- **step-start** ≡ steps(1, start)
- **step-end** ≡ steps(1, end)
- **steps(x [, [start | end]])**
- **cubic-bezier(x, y, z, w)**

Sljedeći isječak koda pokazuje postupan prijelaz u trajanju 4 sekunde iz kvadratnog div elementa visine i širine 60px, koji se nalazi u lijevom dijelu pravokutnika rotacijom za 1080°(odnosno 3 puna kruga) u krug radijusa 30 px koji je pomaknut na desni dio pravokutnika na 3 različita načina. Prvi je *ease*, drugi je *ease-in*, a treći *cubic-bezier*(1.0, -0.5, 0.5, 1). Slika 11. pokazuje njihove položaje u 4 različita trenutka.

```
<style media="screen">
      #container{
                  width:600px;
                  height:450px;
                  border:1px #AAAAAA solid;
                  background-color: #125272;
      }
      .box{
                  font-size:12px;
                  position:relative;
                  width:60px;
                  height:60px;
                  background-color:#105052;
      }
      .box p {
                  text-align:center;
                  padding-top:4px;
      }
      #e1.box {
                  transition: all 4s ease;
      -webkit-transition: all 4s ease;
         -moz-transition: all 4s ease;
             -o-transition: all 4s ease;
                  border:2px #f00 solid;
      }
```

```
      #e2.box {
                  transition: all 4s ease-in;
            -webkit-transition: all 4s ease-in;
               -moz-transition: all 4s ease-in;
                  -o-transition: all 4s ease-in;
                  border:2px #0f0 solid;
      }
      #e3.box {
                  transition: all 4s cubic-bezier(1.0, -0.5, 0.5, 1);
              -webkit-transition: all 4s cubic-bezier(1.0, -0.5, 0.5, 1);
                 -moz-transition: all 4s cubic-bezier(1.0, -0.5, 0.5, 1);
                   -o-transition: all 4s cubic-bezier(1.0, -0.5, 0.5, 1);
                  border:2px #0ff solid;
      }

      #container:hover .box {
                  border-radius:30px;
              -webkit-border-radius:30px;
                 -moz-border-radius:30px;
                  transform: rotate(1080deg);
              -webkit-transform: rotate(1080deg);
                 -moz-transform: rotate(1080deg);
                   -o-transform: rotate(1080deg);
                  -ms-transform: rotate(1080deg);
                     background-color:#fff;
  }
}
</style>

<body>
      <div id="container" class="hover">
            <div id="e1" class="box"> <p> EASE </p> </div>
            <div id="e2" class="box"> <p> EASE-IN </p> </div>
            <div id="e3" class="box"> <p> (1.0, -0.5, 0.5, 1)</div>
      </div>
</body>
```

Slika 11. Primjeri *transition-timing-function*

3.2.2 Animacije

S pojavom CSS3-a možemo stvarati animacije koje mogu zamijeniti animirane slike, Flash ili JavaScript animacije na web stranicama. Animacija se sastoji od 2 komponente: stila koji opisuje CSS animaciju i niza okvira (engl. keyframes) koji određuju početno i konačno stanje animacije, te međustanja.

CSS animacije imaju razne prednosti nad skriptanim animacijama. Lagano je napraviti jednostavne animacije i mogu se stvarati bez prethodnog znanja JavaScripta. Animacije dobro rade i na sustavima koji sporo učitavaju, dok u JavaScript-u čak i jednostavne animacije mogu imati lošu izvedbu ukoliko su loše napravljene. Preglednik sam kontrolira slijed animacija i tako optimizira svoj rad i učinkovitost, npr. ne provodi animacije na tabovima koji nisu otvoreni.

Da bi se stvorio slijed CSS animacija elementu se pridružuje svojstvo *animation* i/ili njegova podsvojstva [6]. Time se podešava trajanje i vremenski trenutak primjene dijela animacije. Animacija se primjenjuje na element koji za vrijednost atibuta *animation-name* svojstvo koje referencira važeće *@keyframes* pravilo. Animacija završava primjenom kombinacije svojstva *animation-duration, animation-iteration-count i animation-fill-mode*.

Podsvojstva atributa *animation* koja se mogu koristiti su:

- **animation-delay**
  Određuje odmak između trenutka kada se učita element i početka primjene niza animacija.

- **animation-direction**
  Određuje da li animacija treba mijenjati smjer na svakom pokretanju niza animacija ili se resetirati na početak i ponavljati.

- **animation-duration**
  Određuje trajanje jednog ciklusa animacije.

- **animation-iteration-count**
  Određuje koliko puta bi se animacija trebala ponavljati, moguće je odrediti da se ponavlja do beskonačnosti.

- **animation-name**
  Opisuje ključne okvire animacije koju određuje *@keyframes* pravilo koje se tako zove.

- **animation-play-state**
  Omogućuje zaustavljanje i nastavak animacije.

- **animation-timing-function**
  Opisuje vremenski tijek animacije, odnosno animirane prijelaze kroz ključne okvire.

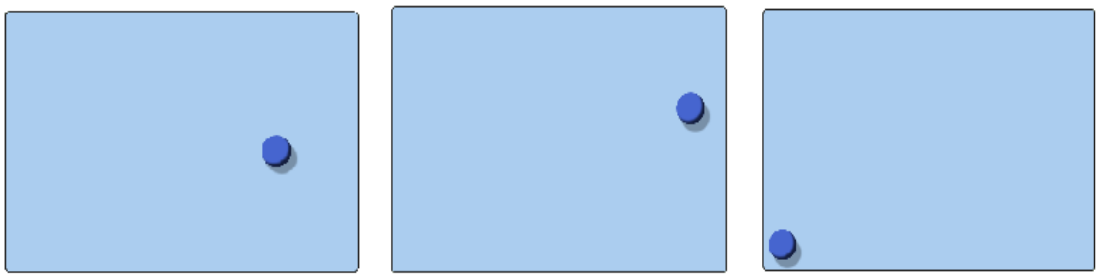
- **animation-fill-mode**
  Određuje koje su vrijednosti dodijeljene animaciji prije i nakon izvršavanja.

@keyframes pravilo sastoji se od ključne riječi @*keyframes*, identifikatora pravila koji daje ime animaciji i niza pravila u vitičastim zagradama koji definiraju animaciju. Svaki ključni okvir određuje se animirani element treba prikazati tijekom trajanja animacije. Ključni okviri koriste postotak kako bi odredili vrijeme kada se određeni dio animacije prikazuje. Zbog njihove važnosti 0% i 100% imaju svoje aliase: *from* i *to*. Sve vrijednosti za ključne okvire moraju biti sortirane rastući po iznosu postotka, a ukoliko se neka preklapaju upotrijebiti će se ono koje je zadnje navedeno.

Dodatnu kontrolu nad animacijama možemo postići uporabom događaja [8] (engl. *animation events*). Time možemo omogućiti interakciju s korisnikom. Događaji se reprezentiraju objektom *AnimationEvent* koji detektiraju kada animacija započinje, završava ili započinje novu iteraciju. Svaki događaj uključuje vrijeme kada se dogodio i ime animacije koja je pokrenula taj događaj.

Sljedeći dio koda pokazuje kako je moguće animirati lopticu koja skače prikazanu na Slici 12. Animacija u smjeru X definirana je s pravilom @keyframes moveX, a u smjeru Y @keyframes moveY. Slijed okvira definiran je u svojstvu *animation* nad elementom <b>.

```
<style media="screen">

      .container b {
            width: 60px; height: 60px;
            border-radius: 70%;
            background-color: #4665cf;
            box-shadow: inset -6px -6px 5px rgba(0,0,0,.7), 10px 15px
                  3px rgba(0,0,0,.3);
            position: absolute;
    -webkit-animation: moveX 2.8s linear 0s infinite alternate, moveY
                  3.3s linear 0s infinite alternate;
      -moz-animation: moveX 2.8s linear 0s infinite alternate, moveY
                  3.3s linear 0s infinite alternate;
      -o-animation: moveX 2.8s linear 0s infinite alternate, moveY 3.3s
                  linear 0s infinite alternate;
            animation: moveX 2.8s linear 0s infinite alternate, moveY
                  3.3s linear 0s infinite alternate;

      @-webkit-keyframes moveX { from { left: 0; } to { left: 640px; }}
      @-moz-keyframes moveX {from { left: 0; } to { left: 640px; }}
      @-o-keyframes moveX {from { left: 0; } to { left: 640px; }}
            @keyframes moveX {from { left: 0; } to { left: 640px; }}

      @-webkit-keyframes moveY {from { top: 0; } to { top: 440px; }}
      @-moz-keyframes moveY {from { top: 0; } to { top: 440px; }}
      @-o-keyframes moveY {from { top: 0; } to { top: 440px; }}
            @keyframes moveY {from { top: 0; } to { top: 440px; }}
      </style>
```



Slika 12. Animacija loptice koja skače

S ovim primjerima možemo vidjeti kako se pomoću CSS-a mogu ostvariti neki zanimljivi efekti na webu. Osim što je nedostatak taj da je, ukoliko želimo da naš primjer radi na svakom pregledniku, potrebno upisivati iste naredbe više puta, ali

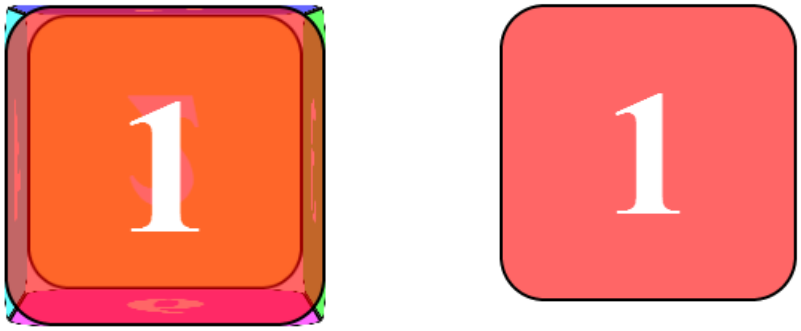
različito prefiksirane, također zasada još ne postoji jednostavan način kako bi se na primjer učitao model objekta iz .obj datoteke i prikazao CSS-om.

Tako na primjer želimo li prikazati kocku potrebno je definirati 6 html elementa i napraviti pripadajuće rotacije i translacije svakog elementa[2].

```
<section class="container">
   <div id="kocka" class="rotiranje">
      <figure
class="naprijed">A</figure>
      <figure class="iza">B</figure>
      <figure class="desno">C</figure>
      <figure class="lijevo">D</figure>
      <figure class="gore">E</figure>
      <figure class="dolje">F</figure>
    </div>
  </section>
```

Primjenom sljedećih transformacija dobit ćemo kocku kao na slici 13. lijevo (ovdje su sada navedene transformacije bez svih prefiksa, kako bi kod bio čitljiviji):

```
#kocka .naprijed  {
       -webkit-transform: translateZ(100px);
}
#kocka .iza {
       -webkit-transform: rotateX(-180deg) translateZ(100px);
}
#kocka .desno {
      -webkit-transform: rotateY(90deg) translateZ(100px);
}
#kocka .lijevo {
      -webkit-transform: rotateY(-90deg) translateZ(100px);
}
#kocka .gore {
      -webkit-transform: rotateX(90deg) translateZ(100px);
}
#kocka .dolje {
      -webkit-transform: rotateX(-90deg) translateZ(100px);
}
```



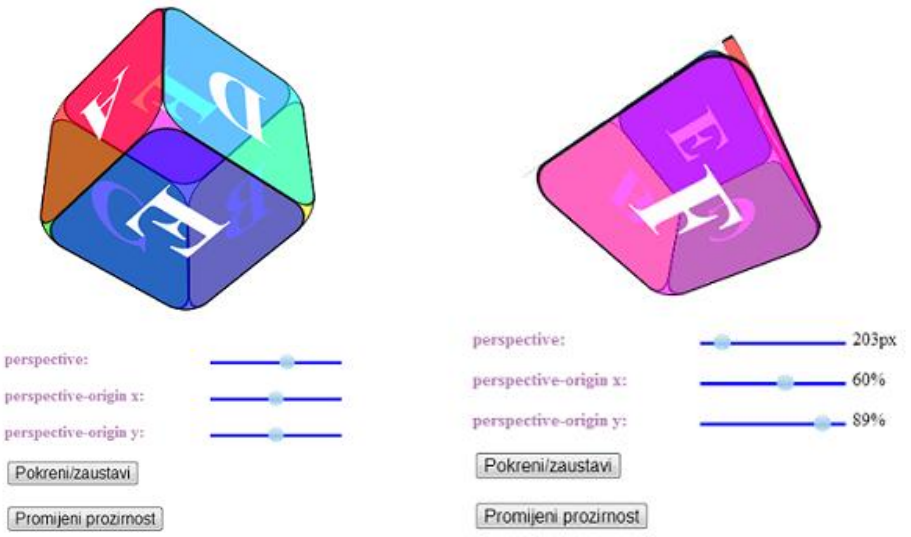
Slika 13. Kocka u CSS-u

Uz kontroliranje događaja pomoću *Javascript*-a može se upravljati CSS svojstvom *backface-visibility* koje kada je postavljeno na vrijednost *hidden* onemogućuje prozirnost objekta, te kocka izgleda kao što je prikazano na slici 13. desno.

Kako bi dodali rotaciju definira se sljedeće *@keyframes* pravilo, te se elementu kocka pridruži trajanje prijelaza te animacija:

```
@-webkit-keyframes spinCubeWebkit {
   0% { -webkit-transform: translateZ(-100px)
                           rotateX(0deg)
                           rotateY(0deg); }
   100% { -webkit-transform: translateZ(-100px)
                             rotateX(360deg)
                             rotateY(360deg); }
 }

# kocka.rotiranje figure{
      -webkit-transition: -webkit-transform 2s;
      -webkit-animation: spinCubeWebkit 10s infinite ease;
}
```

Slika 14. pokazuje lijevo kocku u jednom trenutku rotiranja, dok je desno slika sa izmijenjenim parametrima svojstava *persective*, *perspective-origin-x* i *perspective-origin-y*.



Slika 14. Rotiranje kocke u CSS-u i promjena parametara perspektivne projekcije

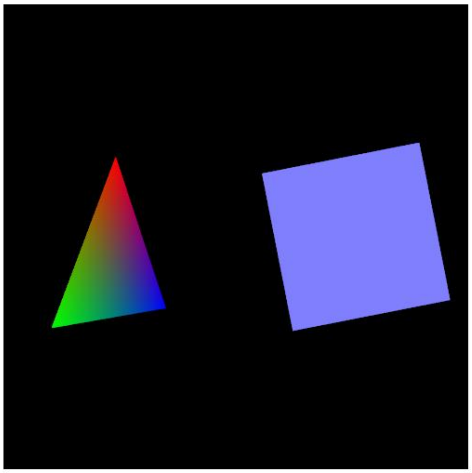
4 WebGL

WebGL (*Web Graphics Library*) je *JavaScript* API za prikazivanje interaktivne 3D i 2D grafike u web preglednicima bez korištenja dodataka temeljen na OpenGL ES 2.0. [10]. WebGL elementi mogu biti pomiješani s drugim HTML elementima te drugim dijelovima stranice ili pozadinom. WebGL programi se sastoje od kontrolnog koda napisanog u J*avaScriptu* i koda za sjenčanje (engl. *shader*) koji se izvršava na grafičkoj jedinici računala (GPU, Graphics Processing Unit). WebGL koristi HTML5 Canvas element i pristupa mu koristeći DOM sučelje. WebGL je podržan u Chrome-u (verzija 8+), Firefox-u (verzija 4+), Safari-ju(verzija 5.1+) i Operi (verzija 12+).

Nedostatak korištenja WebGL-a je taj da za svaki objekt kojeg želimo iscrtati na ekranu postoji podosta manjih postavki koje je potrebno namjestiti što rezultira s dosta koda. Zbog toga su popularne razne biblioteke koje omogućavaju manje posla kod pisanja koda u WebGL-u, npr. biblioteka Three.js olakšava crtanje u WebGL-u.

4.1 Crtanje objekata u WebGL-u

U WebGL-u osnovni element od kojeg se rade objekti je trokut, stoga proces crtanja modela u WebGL-u zahtijeva korištenje *JavaScript*-a za sakupljanje informacija gdje i kako će objekti biti nacrtani.



Slika 15. Trokut i kvadrat u WebGL-u

Osnove programiranja i dobivanja prikaza u WebGL-u pokazat ću kroz primjer koji će iscrtavati trokut i kvadrat, te animirano izvoditi rotaciju nad njima. Na slici 15. je prikaz rezultata koda u jednom trenutku animacije.

4.1.1 Priprema WebGL konteksta

Za početak, potrebno je definirati *canvas* u kojem će se prikazivati objekti koje radimo s WebGL-om i postaviti rukovoditelja događaja kojim će se inicijalizirati WebGL kontekst [12]. U ovom slučaju to je funkcija *webGLStart().* U njoj najprije dohvaćamo referencu na *canvas* element i stavimo je u varijablu *canvas*. Nakon toga pozivaju se funkcije koje će postaviti inicijalizaciju WebGL-a u pregledniku, inicijalizaciju programa za sjenčanje i spremnika, postaviti boju za brisanje (u ovom slučaju crna), omogućiti testiranje dubine i na kraju u ovom slučaju pozvati funkcija *tick()* koja će iscrtavati scenu.

```
//sve što se nalazi u tijelu hmtl dokumenta

<body onload="webGLStart();">
  <canvas id="canvas" style="border: none;" width="500" height="500">
            </canvas>
</body>

<script type="text/javascript">
      function webGLStart() {
        var canvas = document.getElementById("canvas");
              initGL(canvas);   //inicijalizira WebGL
        initShaders();   //inicijalizira shadere
        initBuffers();   //inicjalizira buffere
        gl.clearColor(0.0, 0.0, 0.0, 1.0);
        gl.enable(gl.DEPTH_TEST);
        tick(); //za animaciju

    }

</script>
```

Funkcija *initGL()* prima referencu na canvas i pokušava postaviti webGL kontekst, a ukoliko ne uspijeva javlja upozorenje.

```
function initGL(canvas) {
        try {
            gl = canvas.getContext("experimental-webgl");
            gl.viewportWidth = canvas.width;
            gl.viewportHeight = canvas.height;
        } catch (e) {
        }
        if (!gl) {
            alert("Could not initialise WebGL");
        }
    }
```

4.1.2 Osvjetljavanje scene

Nakon što je WebGL kontekst uspješno postavljen, potrebno je postaviti i programe za sjenčanje. Programi za sjenčanje govore WebGL-u gdje i što treba nacrtati. Funkcija *initShaders()* postavlja programe za sjenčanje.

```
function initShaders() {
   var fragmentShader = getShader(gl, "shader-fs");
   var vertexShader = getShader(gl, "shader-vs");
   shaderProgram = gl.createProgram();
   gl.attachShader(shaderProgram, vertexShader);
   gl.attachShader(shaderProgram, fragmentShader);
   gl.linkProgram(shaderProgram);
   if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {
       alert("Could not initialise shaders");
        }
   gl.useProgram(shaderProgram);
   shaderProgram.vertexPositionAttribute =
            gl.getAttribLocation(shaderProgram, "aVertexPosition");
   gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);
   shaderProgram.vertexColorAttribute =
      gl.getAttribLocation(shaderProgram, "aVertexColor");
   gl.enableVertexAttribArray(shaderProgram.vertexColorAttribute);
   shaderProgram.pMatrixUniform =
      gl.getUniformLocation(shaderProgram, "uPMatrix");
   shaderProgram.mvMatrixUniform =
      gl.getUniformLocation(shaderProgram, "uMVMatrix");
    }
```

Učitava se program za sjenčanje fragmenata (engl. *fragment shader* ili *pixel shader*), koji se dobiva iz skriptnog elementa s identifikacijskim imenom „*shader-fs*“ i program za sjenčanje vrhova (engl. *vertex shader)*, koji se učitava iz elementa „*shader-vs*“. Kreira se program za sjenčanje pozivom funkcije WebGL objekta *createProgram(),* kojemu pričvršćujemo dva programa za sjenčanje i vežemo ih. Nakon toga se provjerava parametar *LINK_STATUS gl* objekta kako bi se vidjelo da li su programi uspješno vezani. Funkcijom *gl.enableVertexAttribArray* naznačuje se WebGL-u da će vrijednosti ovog atributa biti lista u *Javascriptu* (*array)*. I na kraju dohvaćamo dvije uniformne varijable koje predstavljaju matrice.

Funkcija *getShader()* dohvaća program za sjenčanje iz DOM-a, te vraća prevedeni program za sjenčanje ili null, ako on nije mogao biti učitan ili preveden. Kada se pronađe element s traženim id-em, izvuče se njegov sadržaj te ovisno o MIME tipu objekta stvara se program za sjenčanje vrhova (*MIME type "x-shader/x-vertex"*) ili program za sjenčanje fragmenata (*MIME type "x-shader/x-fragment"*).

```
function getShader(gl, id) {
      var shaderScript = document.getElementById(id);
      if (!shaderScript) {
            return null;
      }
      var str = "";
      var k = shaderScript.firstChild;
      while (k) {
            if (k.nodeType == 3) {
                str += k.textContent;
            }
            k = k.nextSibling;
      }
      var shader;
      if (shaderScript.type == "x-shader/x-fragment") {
            shader = gl.createShader(gl.FRAGMENT_SHADER);
      } else if (shaderScript.type == "x-shader/x-vertex") {
            shader = gl.createShader(gl.VERTEX_SHADER);
      } else {
            return null;
      }
      gl.shaderSource(shader, str);
      gl.compileShader(shader);

      if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
            alert(gl.getShaderInfoLog(shader));
            return null;
      }
      return shader;
    }
```

Program za sjenčanje vrhova

Program za sjenčanje vrhova *(eng. vertex shader)* poziva se za svaki vrh. Pozicija vrha se množi s matricom transformacije pogleda i projekcijskom matricom, te se dobivaju koordinate vrha u sceni prikaza. Oznaka *varying* označava da će se vektor boje izmjenjivati za svaki vrh.

```
<script id="shader-vs" type="x-shader/x-vertex">
   attribute vec3 aVertexPosition;
   attribute vec4 aVertexColor;
   varying vec4 vColor;
   uniform mat4 uMVMatrix;
   uniform mat4 uPMatrix;

   void main(void) {
       gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
       vColor = aVertexColor;
  }
</script>
```

Program za sjenčanje fragmenata

Fragment je svaki slikovni element u poligonu. Program za sjenčanje fragmenata određuje boju za svaki slikovni element. *gl_FragColor* je ugrađena GL varijabla koja se koristi za boju fragmenata. Postavljanje vrijednosti u varijablu određuje boju slikovnog elementa.

```
<script id="shader-fs" type="x-shader/x-fragment">
    precision mediump float;
    varying vec4 vColor;
    void main(void) {
       gl_FragColor = vColor;
 }
</script>
```

4.1.3 Kreiranje objekta i primjena boje

Za kreiranje objekata potrebni su nam spremnici u kojima će se nalaziti koordinate vrhova. Inicijalizacija spremnika se obavlja u funkciji *initBuffers().* Funkcija započinje pozivom metode *createBuffer()* gl objekta, koja dohvaća spremnik u koji se kasnije spremaju vrhovi. Spremnik se veže uz kontekst pozivom metode *bindBuffer()*. Nakon toga kreira se *JavaScript* niz koji sadrži koordinate za svaki vrh koji se želi nacrtati. Niz se pretvara u niz WebGL *float* brojeve, te se stavlja u

gl objekt *bufferData()* metodu. Za spremnik se definira broj podataka, te veličina svakog podatka. Isti postupak ponavlja se za spremnik boje. Nakon završetka ove funkcije za korištenje su pripremljeni spremnici koordinata trokuta i kvadrata, te spremnici boja trokuta i kvadrata, redom *triangleVertexPositionBuffer* i *squareVertexPositionBuffer, triangleVertexColorBuffer* i *squareVertexColorBuffer.*

```
function initBuffers() {
   triangleVertexPositionBuffer = gl.createBuffer();
   gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
   var vertices = [
             0.0,  1.0,  0.0,
            -1.0, -1.0,  0.0,
             1.0, -1.0,  0.0
   ];
  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
      gl.STATIC_DRAW);
   triangleVertexPositionBuffer.itemSize = 3;
   triangleVertexPositionBuffer.numItems = 3;
   triangleVertexColorBuffer = gl.createBuffer();
   gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexColorBuffer);
   var colors = [
            1.0, 0.0, 0.0, 1.0,
            0.0, 1.0, 0.0, 1.0,
            0.0, 0.0, 1.0, 1.0
  ];
  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors),
      gl.STATIC_DRAW);
  triangleVertexColorBuffer.itemSize = 4;
  triangleVertexColorBuffer.numItems = 3;
  squareVertexPositionBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
  vertices = [
             1.0,  1.0,  0.0,
            -1.0,  1.0,  0.0,
             1.0, -1.0,  0.0,
            -1.0, -1.0,  0.0
  ];
  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
      gl.STATIC_DRAW);
  squareVertexPositionBuffer.itemSize = 3;
  squareVertexPositionBuffer.numItems = 4;
  squareVertexColorBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexColorBuffer);
  colors = []
  for (var i=0; i < 4; i++) {
      colors = colors.concat([0.5, 0.5, 1.0, 1.0]);
  }
  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors),
gl.STATIC_DRAW);
  squareVertexColorBuffer.itemSize = 4;
  squareVertexColorBuffer.numItems = 4;
}
```

4.1.4. Dodavanje tekstura

Dodavanje tekstura zapravo je poseban način postavljanja "boje točke" 3D objekta [16]. Stoga je potrebno učitati sliku i poslati je programu za sjenčanje fragmenata. Također šalje mu se i podatak koji dio slike iskoristiti za trenutni fragment. Funkcija *initTexture()* imat će sljedeći poziv:

```
var Texture;
  function initTexture() {
    Texture = gl.createTexture();
    Texture.image = new Image();
    Texture.image.onload = function() {
      handleLoadedTexture(Texture)
    }
    neheTexture.image.src = "texture.gif"; //izvor teksture
  }

function handleLoadedTexture(texture) {
  gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
  gl.bindTexture(gl.TEXTURE_2D, texture);
  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
      gl.UNSIGNED_BYTE, texture.image);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER,
      gl.LINEAR);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER,
      gl.LINEAR);

  gl.bindTexture(gl.TEXTURE_2D, null);
}
```

Funkcijom *createTexture()* stvara se referenca na teksturu koja se dohvaća iz izvora i ona se prosljeđuje funkciji *handleLoadedTexture()* koja najprije okrene sliku zbog razlike u koordinatnom sustavu, zatim funkcijom bindTexture() kaže WebGL-u da je to trenutna tekstura, sprema teksturu na prostor na grafičkoj kartici te postavlja linearne parametre za skaliranje.

4.1.5 Crtanje scene

Na kraju funkcije *webGLStart()* pozvana je funkcija *tick().* Njen zadatak je iscrtati scenu na odgovarajući način. Funkcija započinje pozivom metode *requestAnimFrame()* kojim se zahtjeva od JavaScripta da se redovito poziva funkcija *tick()*. Ovisno o pregledniku ova metoda bi se trebala drugačije pozivati, npr. *mozRequestAnimationFrame* za Mozillu, te za Chrome i Safari funkcija se

zove *webkitRequestAnimationFrame*. Ukoliko se uključi skripta *webgl-utils.js* tada je dovoljno pozvati funkciju na ovaj način.

```
function tick()
{
            requestAnimFrame(tick);
            drawScene();
            animate();
 }
```

Nakon što su postavljeni programi za sjenčanje i objekt je konstruiran može se nacrtati scena pozivom funkcije *drawScene()*. Najprije se očisti kontekst u boju pozadine, a zatim odredi perspektiva kamere. Postavi se vidno polje od 45° s omjerom širine i visine *gl.viewportWidth / gl.viewportHeight* (dimenzije canvasa) i odredi da se želi nacrtati predmeti između 0.1 i 100 jedinica. Nakon toga postavi se pozicija objekta tako da se učita matrica identiteta nad kojom se dalje vrše transformacije po potrebi. Nakon što se povežu podaci iz spremnika vrhova i boje, te se oni povežu na kontekst crta se objekt pozivom *drawArrays()* metode.

4.2 Animacije u WebGL-u

Animiranje scene postiže se primjenom lokalnih transformacija kroz vremenski period. Na primjer, ukoliko se želi pomaknuti objekt, recimo trokut, na njega će se primijeniti odgovarajuća lokalna transformacija koja će odrediti njegov položaj, orijentaciju i veličinu. Tu transformaciju vežemo na objekt svaki puta kada se poziva funkcija koja iscrtava objekt. Kako animacija ne bi bila prespora ili prebrza, te kako bi bila neovisna o ciklusu prikazivanja koristi se funkcija *requestAnimFrame* i *Javascript* varijable koje kontroliraju animaciju[13]. Prednost korištenja ove funkcije je da se funkciju koju želimo redovito pozivati, tj. ona koja prikazuje sadržaj, poziva samo u slučaju kada je prozor preglednika fokusiran. Time štedi CPU, GPU i memorijske resurse. Također, funkcija za prikazivanje se poziva onoliko brzo koliko to sklopovlje dozvoljava. Pomoću *Javascript* varijabli kontrolira se animacija i dobiva neovisnost između toga koliko brzo računalo može prikazivati animaciju i onoga koliko brzo se želi da animacija bude prikazivana.

S obzirom da *Javascript* nije višedretveni jezik postoji više različitih strategija kako kontrolirati animaciju i upravljati blokiranjem događaja.

Animacijska strategija

```
//pseudokod
var pocetnoVrijeme  = undefined;
var prosloVremena = undefined;
var brzinaAnimiranja  = 30; //30 ms
function animiraj(deltaT){
    //izracunaj trenutnu poziciju objekta temeljenu na deltaT
}
function onFrame(){
    prosloVremena = (new Date).getTime() – pocetnoVrijeme;
    if (prosloVremena < brzinAnimiranja) return;
      animiraj(prosloVremena);
    pocetnoVrijeme = (new Date).getTime();
}
function zapocniAnimaciju(){
  setInterval(onFrame,brzinaAnimiranja/1000);
}
```

Ovaj način enkapsulira vrijeme koje je prošlo od početka animacije kao varijablu koja kontrolira animaciju. Garantira se da je vrijeme animacije neovisno o tome koliko je puta poziv funkcije zapravo pozvan, a trenutni okvir animacije ovisi o tome koliko je vremena prošlo od početka izvođenja. Nedostatak ove metode je što može doći do izgubljenih okvira (engl. *dropped frames*) [13]. Izgubljeni okvir označava da je objekt trenutno prebačen u novu poziciju na kojoj bi se trebao nalaziti s obzirom na vrijeme koje je prošlo, a međupozicije su zanemarene, pa stoga objekt može skakati po ekranu s jedne pozicije na drugu. Ponekad je ovo ponašanje poželjno, ali na primjer u igrama u kojima trebamo gađati neku pokretnu metu neće se dobiti željeno ponašanje, stoga ova metoda nije dobra u slučajevima kada je, na primjer, potrebno koristiti detekciju sudara.

Simulacijska strategija

```
//pseudokod
var brzinaAnimiranja = 30; //30 ms
var deltaPozicija = 0.1
function animiraj(deltaP){
   // izracunaj trenutnu poziciju objekta temeljenu na deltaT
}
function onFrame(){
    animiraj(deltaPozicija);
}
function pocniAnimaciju(){
    setInterval(onFrame,animationRate/1000);
}
```

Ovim načinom osigurava se neprekinutost prikaza. To je bitno kod detekcije sudara, simulacija vezanih uz fizikalne procese ili igara koje se temelje na umjetnoj inteligenciji. Svakim novim pozivom funkcije osvježava se pozicija objekta. Nuspojava ovog načina je da može doći do smrznutih okvira (engl. *frozen frames*) ako je lista blokiranih događaja dugačka pa se nova pozicija objekta ne izračuna na vrijeme [13].

Tako se u primjeru s trokutom i kvadratom animacija postiže korištenjem animacijske strategije pa se funkcija *tick()* poziva idući puta onda kada preglednik zahtijeva novi okvir koji želi prikazati, a položaj je određen vremenom koje je prošlo od početka animiranja. Varijable *rTri* i *rSquare* koriste se kako bi se pratila rotacija i s vremenom se vrijednosti povećavaju kako bi se objekti rotirali. Animaciju se kontrolira varijablama *lasttime* i *elapsed*.

```
var lastTime = 0;
  function animate() {
    var timeNow = new Date().getTime();
    if (lastTime != 0) {
      var elapsed = timeNow - lastTime;

      rTri += (90 * elapsed) / 1000.0;
      rSquare += (75 * elapsed) / 1000.0;
    }
    lastTime = timeNow;
  }
```

Trokut se rotira za 90° svake sekunde, a kvadrat za 75° svake sekunde.

Sljedeći primjer pokazuje primjenu i kontrolu animacije nad većim brojem objekata. Za inicijalizaciju objekata poziva se funkcija *initWorldObjects()*. U njoj se inicijalizira N objekata.

Slika 16. Tekstura zvijezde

Ovo je konstruktor jednog objekta na sceni, zvijezde:

```
function Star(startingDistance, rotationSpeed) {
    this.angle = 0;
    this.dist = startingDistance;
    this.rotationSpeed = rotationSpeed;
    this.randomiseColors();
  }
```

Zvijezdi se pridružuje početni kut, udaljenost od središta, brzina rotacije i slučajnim odabirom odabire se nijansa boje zvijezde. Svakoj zvijezdi se pri iscrtavanju pridružuje tekstura sa slike 16. na način prikazan u poglavlju 4.1.4.

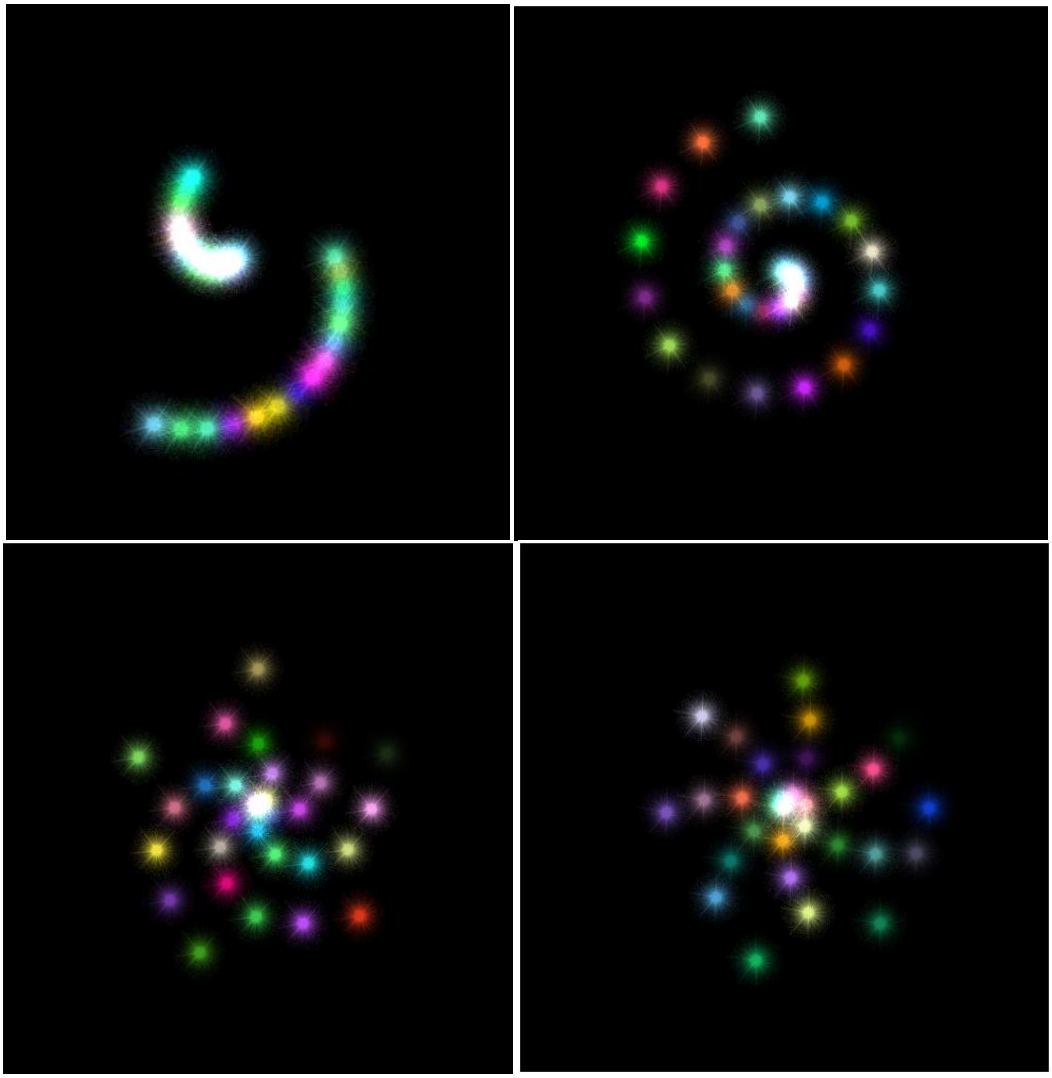
Funkcija koja iscrtava zvijezdu na ekran:

```
Star.prototype.draw = function (tilt, spin) {
   mvPushMatrix();
      // vlasitita rotacija zvijezde
   mat4.rotate(mvMatrix, degToRad(this.angle), [0.0, -1.0, 0.0]);
   mat4.translate(mvMatrix, [this.dist, 0.0, 0.0]);
        // rotiranje da prednja strana bude prema gledatelju
   mat4.rotate(mvMatrix, degToRad(-this.angle), [0.0, -1.0, 0.0]);
   mat4.rotate(mvMatrix, degToRad(-tilt), [-1.0, 0.0, 0.0]);
       //sve zvjezde zajedno rotiraju oko z-osi
  mat4.rotate(mvMatrix, degToRad(spin), [0.0, 0.0, 1.0]);
        // iscrtavanje zvijezde u njenoj boji
  gl.uniform3f(shaderProgram.colorUniform, this.r, this.g, this.b);
  drawStar()
  mvPopMatrix();
};
```

Animacija je napravljena simulacijskom strategijom u kojoj se kontrolira kut rotacije i udaljenost zvijezde od središta.

```
var FPMS = 50 / 1000;
Star.prototype.animate = function (elapsedTime) {
   this.angle += this.rotationSpeed * effectiveFPMS * elapsedTime;
   //ukoliko je položaj zvjezde u centru rotacije osvježi njen položaj
na rub spirale
   this.dist -= 0.05 * effectiveFPMS * elapsedTime;
   if (this.dist < 0.0) {
            this.dist += 5.0;
            this.randomiseColors();
        }
    };
```

Na slici 17. prikazano je nekoliko okvira prilikom pokretanja ovog primjera. Također, omogućeno je pritiskom na strelice na tipkovnici zumiranje i okretanje ravnine rotacije zvjezdica.



Slika 17. Animiranje zvjezdica

S obzirom na činjenicu da je WebGL temeljen je na OpenGL-u mogu se iskoristiti mogućnosti prikazivanja i modeliranja 3D objekata koje su prihvaćene kao standard u računalnoj grafici. Objekt je predstavljen poligonalnom mrežom koja sadrži geometrijske podatke (koordinate točaka), atribute (boje, teksture) i topološke podatke o objektu (povezanost vrhova i bridova).

OBJ (.obj) je datoteka s geometrijskim podacima o 3D objektu [17]. U datoteci se nalaze podaci o koordinatama svakog vrha, koordinatama tekstura, normalama i

pripadnosti vrha određenom poligonu. Podaci u OBJ datoteci zadani su u smjeru suprotnom od kazaljke na satu.

Funkcija *LoadModel(ModelName, CB)* dobiva kao argumente ime datoteke iz koje se želi učitati model, te varijablu CB preko koje će vratiti učitane podatke. Na početku se stvara *XMLHttpRequest* objekt i definira se ponašanje rukovoditelja događaja *onreadystatechange*. Zatim se inicijaliziraju spremnici podataka koji će se učitati.

```
function LoadModel(ModelName, CB){
	var Ajax;
	Ajax = (window.XMLHttpRequest) ? new XMLHttpRequest() : new
			ActiveXObject("Microsoft.XMLHTTP");
	Ajax.onreadystatechange = function(){
		if(Ajax.readyState == 4 && Ajax.status == 200)
		{
			var Script = Ajax.responseText.split("\n");
			var Vertices = [];
			var VerticeMap = [];
			var Triangles = [];
			var Textures = [];
			var TextureMap = [];
			var Normals = [];
			var NormalMap = [];
			var Counter = 0;
			for (var I in Script) {
				var Line = Script[I];
			//....
			}
			CB(VerticeMap, Triangles, TextureMap, NormalMap);
		}
	Ajax.open("GET", ModelName + ".obj", true);
	Ajax.send();
}
```

Nakon učitavanja svih linija i inicijalizacije spremnika slijedi učitavanje podataka red po red (dio koda koji se umeće gore umjesto točkica). Ako je prvi znak u retku "v" radi se o koordinatama vrha, ako je "vt" radi se o teksturama, a ukoliko je "vn" o normalama.

```
if (Line.substring(0, 2) == "v ") {
  var Row = Line.substring(2).split(" ");
  Vertices.push({ X: parseFloat(Row[0]), Y: parseFloat(Row[1]),
      Z: parseFloat(Row[2]) }); }
else if (Line.substring(0, 2) == "vt") {
  var Row = Line.substring(3).split(" ");
  Textures.push({X: parseFloat(Row[0]), Y: parseFloat(Row[1]) });}
else if (Line.substring(0, 2) == "vn") {
  var Row = Line.substring(3).split(" ");
  Normals.push({ X: parseFloat(Row[0]), Y: parseFloat(Row[1]),
      Z: parseFloat(Row[2]) }); }
```

Ako je prvi znak u retku "f" radi se o podacima o poligonu. Podaci o poligonu mogu se sastojati samo od podataka o vrhovima koji pripadaju tom poligonu, ili od podataka o vrhovima i teksturama, ili o vrhovima, teksturama i normalama.

```
else if (Line.substring(0, 2) == "f ") {
   var Row = Line.substring(2).split(" ");
   for (var T in Row) {
         if (Row[T] != "") {
            //ako se redak sastoji od vise vrsta podataka
            if (Row[T].indexOf("/") != -1) {
               var TC = Row[T].split("/");
               Triangles.push(Counter);
               Counter++;

               var index = parseInt(TC[0]) - 1;
               VerticeMap.push(Vertices[index].X);
               VerticeMap.push(Vertices[index].Y);
               VerticeMap.push(Vertices[index].Z);

               index = parseInt(TC[1]) - 1;
               TextureMap.push(Textures[index].X);
               TextureMap.push(Textures[index].Y);

               if (TC.length > 2) {
                  index = parseInt(TC[2]) - 1;
                  NormalMap.push(Normals[index].X);
                  NormalMap.push(Normals[index].Y);
                  NormalMap.push(Normals[index].Z);
                    }
           }
         //inace ucitava samo vrhove
         else {
              Triangles.push(Counter);
                    Counter++;
                    var index = parseInt(Row[T]) - 1;
                    VerticeMap.push(Vertices[index].X);
                    VerticeMap.push(Vertices[index].Y);
                    VerticeMap.push(Vertices[index].Z);
               }
         }
      }
  }
```

Primjer koji slijedi učitava podatke o kućici iz "*House.obj*" datoteke i sprema podatke u 3D objekt. Teksture povezuje sa Slike 18., te nakon toga provodi animaciju okrećući kućicu.

Konstruktor ovakvog 3D objekta sastoji se od:

- podataka o vrhovima, trokutima (poligonim), broju trokuta
- podacima o teksturi, slika koja se primjenjuje kao tekstura
- inicijalna pozicija objekta
- inicijalna rotacija objekta
- inicijalno skaliranje objekta
- kontrolne varijable da li su svi podaci dobro učitani

```
function GLObject(VertexArr, TriangleArr, TextureArr, ImageSrc)
{
    this.Pos = { X: 0, Y: 0, Z: 0 };
    this.Scale = { X: 1.0, Y: 1.0, Z: 1.0 };
    this.Rotation = { X: 0, Y: 0, Z: 0 };
    this.Vertices = VertexArr;
    this.Triangles = TriangleArr;
    this.TriangleCount = TriangleArr.length;
    this.TextureMap = TextureArr;
    this.Image = new Image();
    this.Image.onload = function () {
        this.ReadyState = true;
    };
    this.Image.src = ImageSrc;
    this.Ready = false;
}
```

Slika 18. Teksture za kućicu

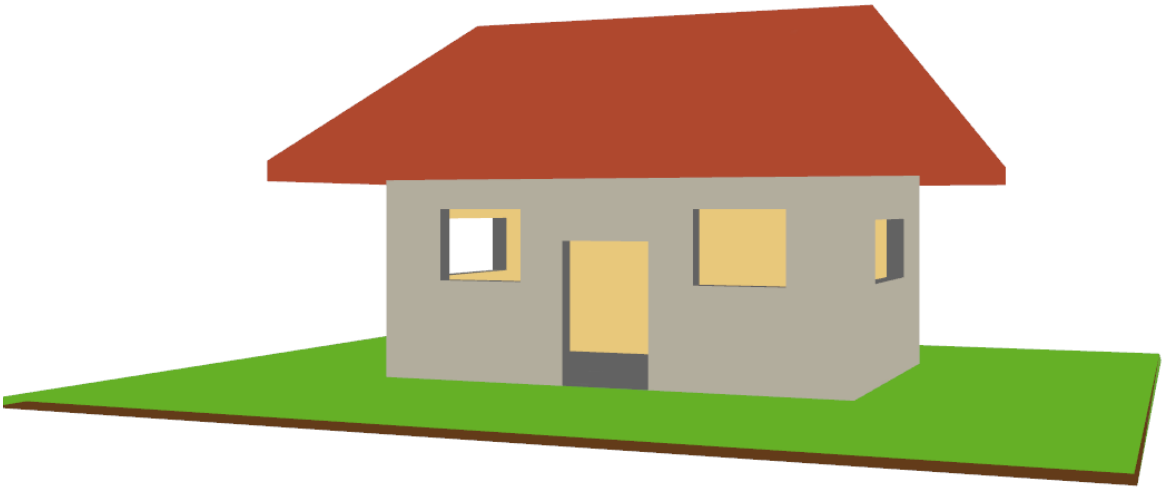
Na slici 19. prikazan je izgled ovako instancirane kućice nakon učitavanja prozora preglednika:

```
function Ready(){
    GL = new WebGL("canvas", "shader-fs", "shader-vs");
    LoadModel("House", function(VerticeMap, Triangles,
TextureMap){
      Building = new GLObject(VerticeMap, Triangles, TextureMap,
"House.png");
      Building.Pos.Z = 650;
      Building.Scale.X = 0.5;
              Building.Scale.Y = 0.5;
      Building.Scale.Z = 0.5;
      Building.Rotation.Y = 180;
      setInterval(Update, 33);
      document.onkeydown = handleKeyDown;
    });
}
```

Moguće je također pomicati objekt (kućicu) po ekranu pritiskom na strelice na tipkovnici.

Animiranje se provodi simulacijskom strategijom:

```
function Update(){
    Building.Rotation.Y += 0.3
    GL.clear(GL.GL.COLOR_BUFFER_BIT|GL.GL.DEPTH_BUFFER_BIT);
    GL.Draw(Building);
}
```



Slika 19. Model kućice

5 Zaključak

Grafika na webu iz dana u dan sve više napreduje, a sve je manja potreba za dodacima web preglednicima kako bi se ostvarili napredniji prikazi. Web programeri više nisu ograničeni na korištenje samo slika ili alternativnih rješenja kao što je Flash. Jedna od velikih prekretnica u 3D grafici na Webu je pojava WebGL-a, koji budući da se temelji na OpenGL-u omogućuje da se u web preglednicima ostvari gotovo sve.

Svim web programerima CSS je jako dobro poznat i vrlo su vješti u njegovom korištenju pri ostvarivanju izgleda web stranice. Najnoviji standard CSS3 donio je nove mogućnosti poput animacija i tranzicija. Iako su mogućnosti velike, animiranje putem CSS-a ipak je ograničeno, naročito kada je riječ o 3D grafici. Međutim onima koji se ne žele baviti kompliciranijim tehnikama prikaza, a žele postići neke efekte koji će privući pažnju ovo je dosta dobro rješenje.

WebGL je međutim puno moćniji, ali zahtijeva i dosta učenja, pogotovo za one koji su se prije toga bavili samo web programiranjem. Za one koji već dobro znaju OpenGL nije potrebno puno vremena da nauče WebGL. Iako je jezik niske razine, te rezultira sa dobrom količinom koda već i za jednostavnije primjere, kompatibilan je preko više preglednika i platformi. Još jedna od prednosti je uska integracija s HTML sadržajem uključujući slojevitu kompoziciju, interakciju s drugim HTML elementima i korištenje standardnih HTML mehanizama za rukovanje događajima. Može se reći da je WebGL budućnost grafike na Webu.

CSS3 donio je ipak mnoštvo načina za oživljavanje sadržaja na web stranici i kada su u pitanju 2D animacije često je sasvim dovoljan. Nasuprot tome, što se prikaza 3D grafike na webu tiče WebGL je u velikoj prednosti. Iako su oba još u razvoju i njihov standard nije u potpunosti definiran zanimljivo je vidjeti kakve sve mogućnosti nude. Kompatibilnost s preglednicama Mozilla, Opera, Safari i Chrome je u CSS podržana kroz različite prefikse, a s obzirom da je WebGL još uvijek u razvoju dohvat konteksta postiže se pozivom *"experimental-webgl".* Iskustvo pri izradi ovog rada i primjera opisanih u radu dalo je zaključak da Google Chrome trenutno najbolje podržava i prikazuje grafiku i animacije na Webu.

6 Literatura

[1] How Web Animation Works
http://computer.howstuffworks.com/web-animation1.htm,
(pristupljeno 15.4.2013.)

[2] CSS Transforms
http://www.w3.org/TR/css3-transforms/,
(pristupljeno 15.04.2013.)

[3] Advanced CSS3 2D and 3D Transform Techniques
http://www.sitepoint.com/advanced-css3-2d-and-3d-transform-techniques/,
(pristupljeno 15.4.2013.)

[4] CSS Transforms
http://css3.bradshawenterprises.com/transforms/,
(pristupljeno 19.4.2013.)

[5] CSS Transitions
http://css3.bradshawenterprises.com/transitions/,
(pristupljeno 19.4.2013.)

[6] Intro to CSS 3D transforms
http://desandro.github.io/3dtransforms/,
(pristupljeno 19.4.2013.)

[7] CSS3 animations
http://www.w3schools.com/css3/css3_animations.asp,
(pristupljeno 23.4.2013)

[8] Using CSS animations
https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Using_CSS_animations?redirectlocale=en-US&redirectslug=CSS%2FTutorials%2FUsing_CSS_animations,
(pristupljeno 25.4.2013.)

[9] Animations
http://msdn.microsoft.com/en-us/library/ie/hh673530(v=vs.85).aspx,
(pristupljeno 7.5.2013.)

[10] WebGL
http://en.wikipedia.org/wiki/WebGL,
(pristupljeno 24.5.2013.)

[11] Getting started with WebGL
https://developer.mozilla.org/en-US/docs/Web/WebGL/Getting_started_with_WebGL?redirectlocale=en-US&redirectslug=WebGL%2FGetting_started_with_WebGL,
(pristupljeno 24.5.2013.)

[12] WebGL Fundamentals
http://www.html5rocks.com/en/tutorials/webgl/webgl_fundamentals/,
(pristupljeno 25.5.2013.)

[13] WebGL: Animating a 3D scene
http://www.packtpub.com/article/webgl-animating-3d-scene,
(pristupljeno 1.6.2013)

[14] Animating objects with WebGL
https://developer.mozilla.org/en-US/docs/Web/WebGL/Animating_objects_with_WebGL,
(pristupljeno 1.6.2013.)

[15] Learning WebGL
http://learningwebgl.com/blog/,
(pristupljeno 8.6.2013)

[16] WEBGL TEXTURES
http://docs.webplatform.org/wiki/tutorials/webgl_textures,
(pristupljeno 10.6.2013.)

[17] Wavefront .obj file
https://en.wikipedia.org/wiki/Wavefront_.obj_file,
(pristupljeno 10.6.2013.)

7 Sažetak

Naslov: Računalna animacija na Webu

U radu su proučene tehnologije na webu koje omogućavaju animacije u web preglednicima. Naglasak je postavljen na CSS3 i WebGL koji omogućavaju izradu animacija bez korištenja dodataka i najbliže su tome da budu u potpunosti podržani u svim web preglednicima. Dan je pregled CSS 2D i 3D transformacijskih funkcija, animiranja pomoću tranzicija i korištenja *@keyframes* pravila za postavljanje animacijskih parametara. Obrađene su osnove korištenja WebGL-a koji omogućava prikaz interaktivne 3D grafike na Webu te se temelji na OpenGL 2.0 ES. Različitim primjerima pokazana je primjena navedenih tehnologija.

Ključne riječi: web grafika, CSS3, animacija, tranzicija, WebGL

8 Abstract

Title: Computer animation on Web

Technologies that allow animation in web browsers have been researched in this work. Emphasis has been placed on CSS3 and WebGL, which enable creation of animations without the use of plug-ins and are the closest to became fully supported in all web browsers. An overview of CSS 2D and 3D transformation functions, animations with transitions and @keyframes rule has been given. The basics of using WebGL, that is based on OpenGL 2.0 ES and enables rendering 3D graphics on Web, is elaborated. The use of this technologies is shown on different examples.

Key words: web graphics, CSS3, animation, transition, WebGL